I/O streams as an introduction to objects and classes

streams and basic file I/O

Input can come from the keyboard or from a file
Output can go to the screen or to file

- Input and output is delivered to and from your program via a C++ construct known as a stream
- Streams are the first examples of **objects**

Object – a special kind of variable that has its own special-purpose functions
The ability to handle objects sets C++ apart from earlier programming languages

Stream – A flow of data. If the flow is into your program it is an input stream and out of your program it is an output stream.

```
#include <iostream> //name of the header file for i/o
```
must be included in your program:

cin is an input stream connected to the keyboard
cout is an output stream connected to the screen

```
#include <fstream> //name of the header file for file i/o
using namespace std; //for ifstream and ofstream
```

You can define other streams that come and go from files and use it the same way you use cin and cout

```
//header comments
//directives
#include <iostream> //name of the header file for i/o
#include <fstream> //name of the header file for file i/o

int main()
{
	int num1, num2, num3, num4;

	using namespace std; //for ifstream and ofstream
	ifstream inStream;
	ofstream outStream;

	//connecting to the input file:
	inStream.open("inputData.txt");

	if (inStream.fail())
		//did not find the file
		cout << "Input file did not open correctly" << endl;
	else
	{
		inStream >> num1 >> num2 >> num3 >> num4;

		//connecting to an output file:
		outStream.open("outputData.txt");

		outStream << "The sum of the four numbers is "
			<< (num1 + num2 + num3 + num4) << endl;

		inStream.close();
		outStream.close();
	}
	return  0;
}
```

Introduction to classes and objects

Object – A variable that has functions as well as data associated with it

cin
cout
inStream (see above)
outStream (see above)

the functions open and close are associated with them
The two open functions are not the same, one is for reading, one is for writing to files

are all examples of objects

member function – a function that is associated with an object

```
using namespace std; //for ifstream and ofstream
ifstream inStream, in; //two input stream objects
ofstream outStream, out; //two output stream objects

//connecting to the input file:
inStream.open("inputData.txt");
in.open("inFile.txt")

//connecting to an output file:
outStream.open("outputData.txt");
out.open("outFile.txt");
```

class – a type whose variables are objects

the class ofstream has a member function called precision, this function does not belong to ifstream:

```
outStream.precision(2); //same as cout
```

dot operator
calling object

```
//function call
calling_object.member_function_name(argument list);
```

The exit statement:

```
#include <cstdlib> //for exit(1)
using namespace std; //for exit(1)

if (inStream.fail())//fail is a member function
{
	//did not find the file
	cout << "Input file did not open correctly" << endl;
	exit(1);
}
```

When reading data from files you should not include prompts

Appending to a file:

```
ofstream outstream;
outStream.open("outputData.txt", ios::app);
```

File name as input:

```
//header commments
//directives
#include <iostream> //name of the header file for i/o
#include <fstream> //name of the header file for file i/o
#include <cstdlib> //for exit()

int main()
{
	int num1, num2, num3, num4;

	using namespace std; //for ifstream and ofstream
	ifstream inStream;
	ofstream outStream;

	//connecting to the input file:
	inStream.open("inputData.txt");

	if (inStream.fail())
	{
		//did not find the file
		cout << "Input file did not open correctly" << endl;
		exit(1);
	}
	inStream >> num1 >> num2 >> num3 >> num4;

	char filename[16];

	//user should enter filename with .txt extension
	cout << "Enter the filename: ";
	cin >> filename;
	//connecting to an output file:
	outStream.open(filename, ios::app);

	outStream << "The sum of the four numbers is "
		<< (num1 + num2 + num3 + num4) << endl;

	inStream.close();
	outStream.close();
	return  0;
}
```

6.2 tools for stream I/O

Output **format** – layout of the program's output

The **magic formula**:

```
	ofstream outStream;
	char filename[16];

	//user should enter filename with .txt extension
	cout << "Enter the filename: ";
	cin >> filename;
	//connecting to an output file:
	//use app if you want to append data to the file
	outStream.open(filename, ios::app);

	//magic formula
	//setf - set flags
	outStream.setf(ios::fixed); //fixed – usual way, i.e not e-notation
	outStream.setf(ios::showpoint); //show the decimal point
	outStream.precision(2);
```

Formatting Flags for `setf`

Flag	Meaning	Default
`ios::fixed`	If this flag is set, floating-point numbers are not written in e-notation. (Setting this flag automatically unsets the flag `ios::scientific`.)	Not set
`ios::scientific`	If this flag is set, floating-point numbers are written in e-notation. (Setting this flag automatically unsets the flag `ios::fixed`.) If neither `ios::fixed` nor `ios::scientific` is set, then the system decides how to output each number.	Not set
`ios::showpoint`	If this flag is set, a decimal point and trailing zeros are always shown for floating-point numbers. If it is not set, a number with all zeros after the decimal point might be output without the decimal point and following zeros.	Not set
`ios::showpos`	If this flag is set, a plus sign is output before positive integer values.	Not set
`ios::right`	If this flag is set and some field-width value is given with a call to the member function `width`, then the next item output will be at the right end of the space specified by `width`. In other words, any extra blanks are placed *before* the item output. (Setting this flag automatically unsets the flag `ios::left`.)	Set
`ios::left`	If this flag is set and some field-width value is given with a call to the member function `width`, then the next item output will be at the left end of the space specified by `width`. In other words, any extra blanks are placed *after* the item output. (Setting this flag automatically unsets the flag `ios::right`.)	Not set

Manipulator – a function that is placed after the insertion operator <<
We have already seen one Manipulator << endl

```
//header comment here
#include <iostream> //standard library for i/o
#include <iomanip> //for manipulators, setprecision,
using namespace std;

int main()
{
	int hours;
	double grossPay, rate;

	cout << "Enter the hourly rate of pay: $";
	cin >> rate;

	cout << "Enter the number of hours worked,\n"
		<< "rounded to a whole number of hours: ";
	cin >> hours;

	grossPay = (rate * 40);
	//set the number of decimal places for doubles
	cout.setf(ios::fixed);
	cout.setf(ios::showpoint);
	//cout.precision(2); //use any number her for the number of decimal places

	cout << "\nHours worked (" << hours << ") the first 40 hours at $";
	cout << setprecision(2) << rate << " per hour" << endl;
	cout << "\nTime and a half rate is $" << setprecision(2) << (1.5 * rate) << endl;
	cout <<  setprecision(2) << "\nGross pay is $" << grossPay << endl << endl;

	return  0;
}
```

6.3 character I/O

Member functions get and put

The member function **get** allows you to read in one **character** at a time
When you use the extraction operator >> skipping blanks is done automatically, with get a new line or blank character may be input.
The member function **put** allows you to output one character

```
char letter;
cin.get(letter);//can also be used with ifstream

char letter = 'X';
cout.put(letter);//can also be used with ifstream
```

NOTE: If you want your program to read an entire line of input (with white space included) use the function **getline**

The eof member function

Eof stands for end of file

Streams as arguments to functions:

The formal parameter MUST be call by reference

```
//Precondition: The file streams Messy and neat have been connected and opened
//Postcondition: The numbers in the messy file have been corrected and
//display in both the neat file and the screen
void MakeNeat(ifstream& messy, ofstream& neat, int num, int  fieldwidth)
{
	neat.setf(ios::fixed);
	neat.setf(ios::showpoint);
	neat.setf(ios::showpos);
	neat.precision(num);

	cout.setf(ios::fixed);
	cout.setf(ios::showpoint);
	cout.setf(ios::showpos);
	cout.precision(num);

	double next;
	messy >> next; //initialization
	//cout << next;
	while (!messy.eof())//there is another number to read
	{
		cout << setw(fieldwidth) << next << endl; //onto screen
		neat << setw(fieldwidth) << next << endl; //into the neat file
		messy >> next; //update
	}

}
```