Review: Data Files

Data File − an external collection of related data treated as a unit (such as hourly precipitation data, daily temperature values, computer science text books, etc.) Data can be created separately from a program, stored in some auxiliary storage devices, and accessed by programs when needed.

Text Data File	– consists of characters (also called ASCII files) – data must be converted to internal formats – are organized around lines – are created and used by text editors and/or programs
Binary Data File	– consists of internal computer format – no format conversion is necessary , therefore they are much faster to input/output – are organized around records – are created and used by programs only

Header File – needed for C++ data files processing

```
#include <fstream>
```

Create a File Object

```
     ifstream inFile;      // input file stream: to read from
     ofstream outFile;     // output file stream: to write to
```

Open a Data File – using the `open` member function of the file stream object

```
     inFile.open( "Scores.txt");
     outFile.open( "C:\\data\\Grades.txt");
```

The `open` member function searches for a file and if found it prepares it for processing; the function has one argument: the name of the data file, including the extension (it could also include a path)

- Open a file for reading – the data file must be created first; you always start reading a file from the beginning. It is an error to try to open a file that does not exist (for instance when you misspell its name).
- Open a file for writing – if there's no such file, it is created; if there is a file with this name, it is erased!

There are ways to check if the `open` member function successfully opened the file. This is the function that "connects" the file object in your program with the actual data file. Once that the connection is established, you no longer need the name of the actual file: the other functions that need to deal with the file will use the name of the file object.

Create and Open File Object at the same time (in one line of code)

```
     ifstream inFile("Scores.txt");
     ofstream outFile("C:\\data\\Grades.txt");
```

Open a File Using a String Object

```
ifstream inFile;
string   fileName;

cin >> fileName;
inFile.open( fileName);    // works in C++ 11

inFile.open( fileName.c_str());
                           // works in older versions of C++
```

Test for File Open Errors

```
inFile.open( "Scores.txt");
if ( !inFile )  // treat the file stream object as a Boolean expression
{
    cout << "Error opening Scores.txt for reading");
    exit(101);  //  101 represents an error code
                //  it could be any number of your choice
}
```

Another way to test for file open errors is to use the `fail` member function of the file stream object:

```
inFile.open( "Scores.txt");
if ( inFile.fail() )
{
    cout << "Error opening Scores.txt for reading");
    exit(EXIT_FAILURE); //  use this named constant instead of
                        //  a literal
}
```

Close a Data File – using the `close` member function of the file stream object

```
inFile.close();
outFile.close();
```

The `close` member function terminates the "connection" established by `open` between your program and the actual data file. What happens if you forget to close a file you no longer need? Sometimes nothing happens: at the end of the program all files are closed automatically. However, it is a good practice to close all files, to avoid wasting resources (that are needed to keep those files open).

Using `exit()` when opening/closing a file is the recommended approach at this level. To use `exit()` and `EXIT_FAILURE` include `<cstdlib>`.

Write Data to a Text File – using <<
Data in internal format are converted to strings of characters and sent to the specified file (it is similar to using << and the cout object)

```
cout    << year << " " << price << endl;
outFile << year << " " << price << endl;
```

File output may be formatted the same way the screen output is formatted.

```
outFile << setprecision(2) << fixed;
outFile << setw(5) << year
        << setw(8) << price << endl;
```

Read Data from Text File – using >>
Read and convert a stream of characters from the input file into internal format and store these values in the list of specified variables (it is similar to using >> and the cin object)

```
cin    >> year >> price;
inFile >> year >> price;
```

When reading data from the keyboard extensive validation is required; when reading data from a file, we assume data have been validated, therefore we may consider that the file is valid and correctly formatted.

Use the End Of File marker to stop reading
In most of the cases we do not know the number of items we have to read from a file, therefore the reading will stop when the end of the file is encountered.

Example: Read a list of temperature values and find their average.

```
ifstream inFile("temperatures.txt");
avg = count = 0;
while (inFile >> temp )     // get the next value from the input file
{
    avg += temp;            // add current temp value to avg
    count++;                // keep track of the number of values
}
if ( count != 0 )           // never divide by 0!
{
    avg /= n;
    cout << count << " " << avg << endl;
}
```

The expression **inFile >> temp** evaluates to true if an item is successfully read from the file. At the end of the file there's nothing left to read, therefore the expression will evaluate to false and the loop will stop.

File Stream Objects as Function Arguments – a file stream object must be a reference parameter

Example:
```
int readOneNumber( ifstream &inFile)
{
    int number;
    inFile >> number;
    return number;
}
```

Other I/O Member Functions for File Stream Objects

```
getline    // get a string including white spaces
get        // get one character
```

Working with Characters and Strings

```
ifstream myFile("Cities.txt");
string city;


myFile  >> city;   // San Francisco
cout << city;      // San      (stop reading at a white space)


getline(myFile, city);   // San Francisco  (reads the entire line)
cout << city;            // San Francisco



char ch1, ch2;
cin  >> ch1 >> ch2;   // A     B  (cin skips leading white spaces)
cout << ch1 << ch2;   // AB


char ch1, ch2;
ch1 = myFile.get();        // A     B
ch2 = myFile.get();        // myFile.get()  does not skip white spaces
cout << ch1 << ch2;        // A  (ch2 contains a space character)
```

Mixed Input

Assume the input file has the following format: a number on one line, a city name on the next line:

```
2017
San Francisco
```

```
ifstream myFile("Cities.txt");
int year;
string city = "";

myFile  >> year;          // 2017
getline(myFile, city);
cout << year; // 2017
cout << "[" << city << "]"; // []
```

What does `city` contain? You've pressed "ENTER" after entering the year in the input file: this prevented getline() to do its job: getline() reads everything and stops at the first newline that encounters. To ignore this unwanted newline we must use ignore():

```
myFile  >> year;          // 2017
myFile.ignore();
getline(myFile, city);
```

To ignore (skip) the next 15 characters or until a newline is encountered:

```
myFile.ignore(15, '\n');
```

Another way to call the `getline` member function:

```
getline(myFile, city, '\n'); // '\n' is the default delimiter
getline(myFile, city, ':');  //  use ':' as a delimiter
```