Files

In this lesson you will learn how to create and read files.
There are two types of files that we will look at: text files and binary files.

Text files have variable length records. (One line of text may have 20 characters, the next line 5, etc.) A text file must be read sequentially: line 1, then line 2, etc. Text files are also called Sequential files.

Each record in a binary file must be exactly the same size. Because all of the records are the same length, a binary file can be read either sequentially or randomly. Suppose each record has exactly 20 bytes. The location of the first record in the file is always known. The second record begins 20 bytes (1 * 20) past that. The fifth record begins 80 bytes (4*20) past the beginning of the file. If the position where each record starts can be calculated, then we can read the Nth record without reading all of the records before it. Binary files are also called Random Access files.

Text files, or sequential files are easier to use. We will look at them first.

Let's suppose that you have written a program to print a table of inches and centimeters.

```
//Print table of inches to CM
#include <iostream.h>
void main(void)
{  cout<<"INCHES\tCM\n"; //print heading
   for(int inches=1; inches<20; inches++)
     cout<<inches<<"\t"<<inches*2.54<<"\n";
} //main
```

The table that is printed looks like the one shown below.

```
INCHES  CM
1       2.54
2       5.08
3       7.62
4       10.16
5       12.7
6       15.24
7       17.78
8       20.32
9       22.86
…
19      48.26
```

Output Files

Now, let's suppose that you would like to include the table in a Word document, or you want to print it out and carry it around with you. There is no easy way to copy the table into the word document the way it is now.

Solution: Instead of printing the table on the screen we will print it to a file. The statement to print to a file is exactly the same as printing on the screen:

```
cout<<"Hello"; //sends the word Hello to cout (the screen)
abc<<"Hello"; //sends the word Hello to a file named abc.
```

Unfortunately, there is a bit more, because cout is always there. If we want to print to a file we have to declare the file and open it. Opening the file associates a system name, such as "a:\people.txt" with the name used by the program. When the program tries to write to the A:

drive errors could occur, such as there is no disk in the A: drive, or the disk is full. So, we also have to check for errors.

The program below prints the same table to a file.

```
    //Write table of inches to CM
 1  #include <fstream.h>
    void main(void)
 2  {  ofstream inch;  //output file stream
 3     inch.open("c:/inches.txt"); //associates name inch with file
 4     inch<<"INCHES\tCM\n"; //print heading
       for(int inches=1; inches<20; inches++)
 5       inch<<inches<<"\t"<<inches*2.54<<"\n";
 6     inch.close(); //close anything you open!
 7     cout<<"File created\n"; //cout is used to inform user
       } //create table
    } //main
```

Explanation:

1. **fstream.h** has everything that is in **iostream.h** plus support for files. **fstream.h** *replaces* **iostream.h** when you are using files.
2. ofstream is a new type. Just like variables can be declared as int or char, declaring something as ofstream means that it is a file object. The object inch is an output file. (cout is also an output file. cin is an input file)
3. An output file has several different methods. Methods are similar to functions except that they belong to the object. So instead of calling a function open, we say inch.open. The system name for the file is passed as a string to the open method. After inch is opened, we can print to inch just like we would print to cout.
4. "INCHES", tab, "CM", and a line feed are sent to inch. This is printed to the file that inches refers to exactly the same way it would if the statement has said cout<<.
5. Statement 5 is in side the loop, so it prints each line of the table to the file.
6. Any file that is opened must be closed! "The close method does not have any arguments.
7. Notice that cout can still be used to print a message on the screen. Without this statement the user would not see any output at all.

After you run this program, go to the C:\ drive and double click on inches.txt. You will see that it looks exactly like the one that is printed on the screen in the first program. Once it is in a file you can copy, paste, print, whatever you want.

Experiment: Create a table showing how much tip to leave. Just modify the inches program so that it shows cost and 15%, then add another column showing 20%.

The example below is identical except that the file is output to the A:\ drive. This presents an additional problem because we can not assume that there is a disk in the A: drive. We have to check for errors.

```
   //Write table of inches to CM
   #include <fstream.h>
   void main(void)
   {  ofstream inch;  /output file stream
      inch.open("A:/inches.txt"); //associates name inch with file
1     if (inch.fail()) //maybe no disk in A:, or it is full
2       cout<<"File error\n"; //cout to notify user of error
3     else //create table by writing to inch
4     { inch<<"INCHES\tCM\n"; //print heading
        for(int inches=1; inches<20; inches++)
          inch<<inches<<"\t"<<inches*2.54<<"\n";
        cout<<"File created\n"; //cout is used to inform user
        inch.close(); //close anything you open!
5     } //create table
   } //main
```

Explanation:

1. After trying to open the file we use the fail method to make sure that the file opened successfully before we try to write to the file. The fail method does not take any arguments, and returns true if the open fialed, false if it opened successfully.
2. If the file did not open, we use cout to print an error message.
3. If the file opened, statements in the else block create the table just like the previous program.
4. The close method is called inside the else block. If the file open failed, we do not do anything else to the file.

Input Files

The program below inputs numbers (from the user) and prints the total when the user enters 0 or a negative number.

```
//Read in numbers until 0 is entered, print total
#include <iostream.h>
void main(void)
{  int num, total=0;
   cout<<"Enter a number:";
   cin>>num;
   while (num>0)
   { total+=num;
     cout<<"Enter a number:";
     cin>>num;
   } //while
   cout<<"Total="<<total<<"\n";
} //main
```

The input comes from the user so cin is used. Whenever cin is used, we also include a prompt so the user knows what to type. When we read from a file we will not need the prompts.

Before writing the program use notepad to create a file with a few numbers and save it as c:/numbers.txt. The numbers can be separated by any kind of white space: spaces, line feed, or tabs:

```
12 34
66 5
10
500
```

The program below reads the numbers from the file instead of using cin. Notice that prompts are not necessary. Instead of looking for a zero, the loop continues until the end of the file EOF is reached.

```
  //Read file of numbers, find total
1 #include <fstream.h>
  void main(void)
2 {  ifstream numbers;  //input file stream
3    numbers.open("c:/numbers.txt"); //associates numbers with file
     int num, total=0;
4    while (!numbers.eof()) //eof=end of file
5    {  numbers>>num; //reads a number from the file
        cout<<num<<"\n"; //print each number
        total+=num;
     } //while not end of file
     cout<<"Total="<<total<<"\n";
6    numbers.close();
  } //main
```

Explanation:

1. The header file fstream.h is used instead of iostream.h.
2. numbers is declared as an input stream file.
3. numbers is opened. WE ARE NOT CHECKING FOR AN ERROR!!!
4. The loop ends when the end of the file is reached.
5. Instead of cin>>num, the statement has been changed to read from the file called numbers instead.
6. Always close anything you open!

This program is really "plying with fire!" Assuming that a file called numbers.txt exist on the C: drive is very risky! A better verwsion is shown below:

```
  //Read file of numbers, find total
  #include <fstream.h>
  void main(void)
  {  ifstream numbers;  //input file stream
     numbers.open("c:/numbers.txt"); //associates with a file
1    if (numbers.fail())
     { cout<<"File not found\n";
      } //failed
2    else
     {   int num, total=0;
         while (!numbers.eof()) //eof=end of file
         {  numbers>>num; //reads a number from the file
```

```
           cout<<num<<"\n"; //print each number
           total+=num;
        } //while not end of file
        cout<<"Total="<<total<<"\n";
        numbers.close();
3    } //file opened and created
} //main
```

The lines shown in bold were added. The file is read in #2 through #3.

Passing a file to a function

Instead of having the whole program in main, we would like to write a separate function to read the file.

```
//Read file of numbers, find total
#include <fstream.h>
1 void readNumbers(ifstream nFile)
{ int num, total=0;
  while (!nFile.eof()) //eof=end of file
  {  nFile>>num; //reads a number from the file
     cout<<num<<"\n"; //print each number
     total+=num;
  } //while not end of file
  cout<<"Total="<<total<<"\n";
  nFile.close();
2 } //readNumbers: file opened and created

void main(void)
{  ifstream numbers;  //input file stream
   numbers.open("c:/numbers.txt"); //associates with a file
   if (numbers.fail())
     cout<<"File not found\n";
   else
3    readNumbers(numbers);
} //main
```

Explanation: Lines 1 to 2 is a function that read the file, and prints the total. It is passed the file that was successfully open as an argument.
Function main is much simpler now: After main tries to open the file it will either print the error message, or call function `readNumbers` and passes the file to it.

If you want to find the largest and smallest, you can modify readNumbers to read the first number before the loop:

```
void readNumbers(ifstream nFile)
{ int num, total=0;
  nFile>>num; //reads first number from the file
  total=num;
  while (!nFile.eof()) //eof=end of file
  {  nFile>>num; //reads a number from the file
     cout<<num<<"\n"; //print each number
     total+=num;
  } //while not end of file
  cout<<"Total="<<total<<"\n";
  nFile.close();
} //readNumbers: file opened and created
```

Experiment: Add the rest of the code to find the smallest and largest.

Reading Names

The next program will read name and year born from a file. It will print a list of names and ages. Before running this program create a file of people as shown below. Save it as A:/names.txt.

```
Jay 1974
Debbie 1976
John 1918
Greg 1954
Carol 1999
Sam 2000
```

Important: The program will look for a name and an integer, make sure that the date in the file is arranged in exactly the same order! The function to read the file is called from main just like in the previous program.

```
#include <fstream.h>
void readfile(ifstream people)  //passed from main
{ char name[15];
  int yearborn, age;
  cout<<"NAME\tAGE\n";
  people>>name; //read first before loop!
  while (!people.eof()) //eof=end of file
  {  people>>yearborn;
     age=2001-yearborn;
     cout<<name<<"\t"<<age<<"\n"; //print each number
     people>>name; //read next name
  } //while not end of file
  people.close();
} //readfile

void main(void)
{  ifstream names;  //input file stream
   names.open("c:/names.txt"); //associates with a file
   if (names.fail()) //maybe file does not exist
     cout<<"File not found\n"; //cout notifies user of error
```

```
   else
     readfile(names);
} //main
```

Experiment: Ask the user what year it is (ask just once!). Use that year instead of 2001. Find the average age, the youngest, and the oldest.

Read into an Array

The next program will read name and year born from a file, and put directly into an array. It then calls a function to print the names from the array. Notice that readfile returns the number of people read. This is important because although the size of the array if 20, when we process it, we want to only do the number of people actually read.

```
   //Read file of name and year born. Print name and age
   #include <fstream.h>
1  int readfile(ifstream pFile,char names[][15],int years[])
   { int count=0;
     pFile >>names[count]; //read first before loop!
     while (!pFile.eof() && count<20) //eof=end of file
     {  pFile >>years[count];
        count++;
        if (count<20) pFile >>names[count]; //read next name
     } //while not end of file
     pFile.close();
     return count;
   } //readfile

2  void printAll(char names[][15],int years[], int count)
   { int age;
     cout<<"NAME\tAGE\n";
     for(int i=0; i<count; i++)
     { age=2001-years[i];
       cout<<names[i]<<"\t"<<age<<"\n";
     } //for loop
   } //printAll

3  void main(void)
   {  ifstream names;  //input file stream
      char people[20][15]; //allow 20 people, 14 letters for name
      int year[20]; //yearborn for same 20 people
      int numPeople; //number of people, there may be less than 20
      names.open("c:/names.txt"); //associates with a file
      if (names.fail()) //maybe file does not exist
        cout<<"File not found\n"; //cout to notify user of error
      else
      {  numPeople=readfile(names, people,year);
         printAll(people, year, numPeople);
      } //read file and print
   } //main
```

There are 3 functions: main, printAll, and readfile. Function main tries to open the file. If it opens successfully, main calls readfile and passes the file, the array of names and the array of years. These arrays are local to main and do not have any values in them yet.

Readfile reads the file until it comes to the end of the file, or when it gets to 20. There is room for only 20 names. After reaching eof, function readfile returns count. If there are only 5 names in the file, we want to print just 5, not the 20 that we declared space for,

After readfile returns that number of names read in, main calls function printAll and passes the two arrays and count.

Experiment: Turn main into a menu driven program. Choices could include find youngest, find oldest, printAll, etc.