Chapter 12 – File Operations

12.1 What is a File?

- A file is a collection on information, usually stored on a computer's disk. Information can be saved to files and then later reused.

12.2 File Names

- All files are assigned a name that is used for identification purposes by the operating system and the user.

Table 12-1

File Name and Extension	File Contents
MYPROG.BAS	BASIC program
MENU.BAT	DOS Batch File
INSTALL.DOC	Documentation File
CRUNCH.EXE	Executable File
BOB.HTML	HTML (Hypertext Markup Language) File
3DMODEL.JAVA	Java program or applet
INVENT.OBJ	Object File
PROG1.PRJ	Borland C++ Project File
ANSI.SYS	System Device Driver
README.TXT	Text File

12.3 Focus on Software Engineering: The Process of Using a File

- Using a file in a program is a simple three-step process
  - The file must be opened. If the file does not yet exits, opening it means creating it.
  - Information is then saved to the file, read from the file, or both.
  - When the program is finished using the file, the file must be closed.

Figure 12-1

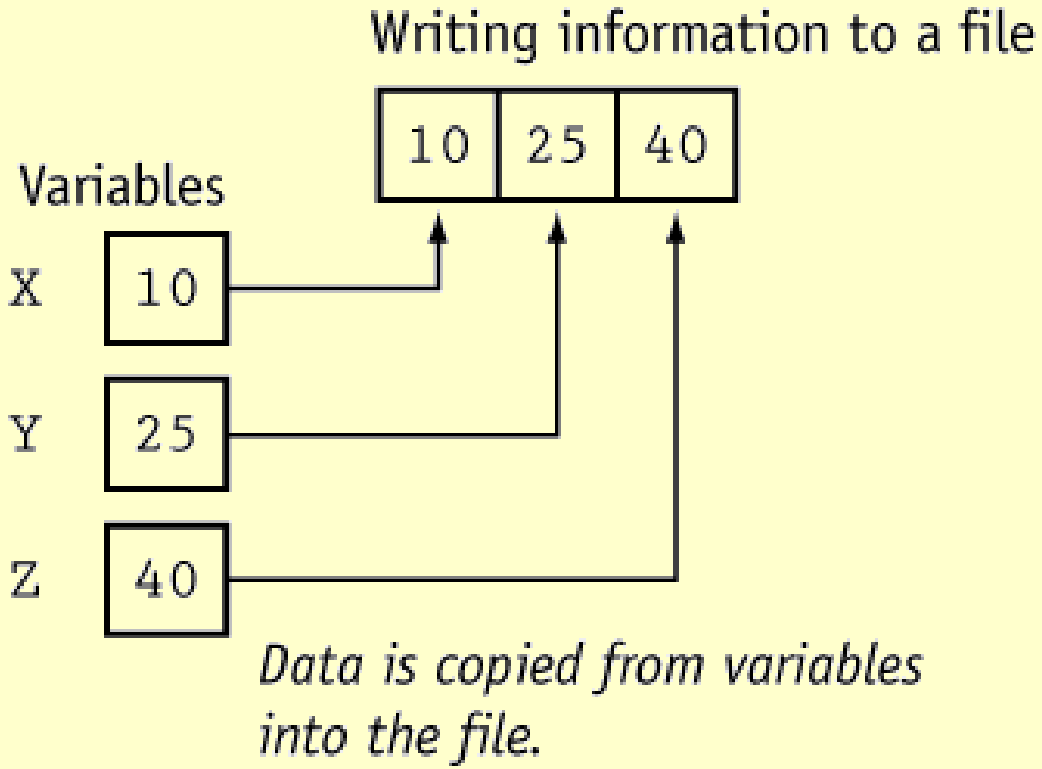
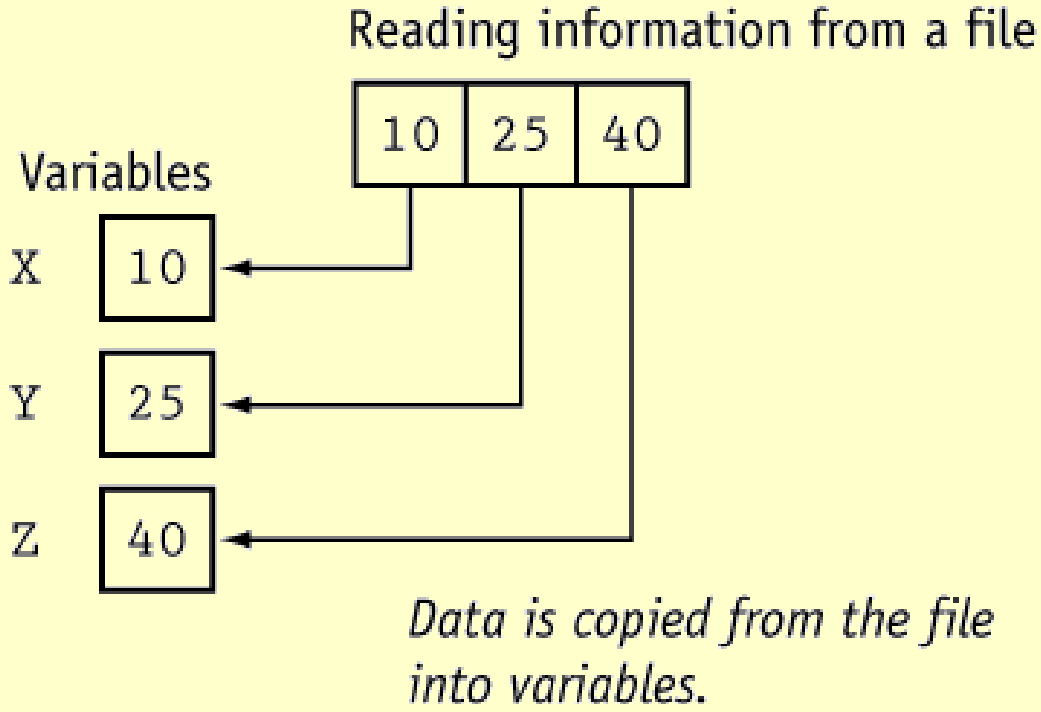

Writing information to a file
10
25
40
Variables
X
10
Y
25
Z
40
Data is copied from variables into the file.

Figure 12-2


Reading information from a file
10
25
40
Variables
X
10
Y
25
Z
40
Data is copied from the file into variables.

12.4 Setting Up a Program for File Input/Output

- Before file I/O can be performed, a C++ program must be set up properly.
- File access requires the inclusion of fstream.h

12.5 Opening a File

- Before data can be written to or read from a file, the file must be opened.

```
ifstream inputFile;
inputFile.open("customer.dat");
```

Program 12-1

```
// This program demonstrates the declaration of an fstream
// object and the opening of a file.
#include <iostream.h>
#include <fstream.h>

void main(void)
{
    fstream dataFile; // Declare file stream object
    char fileName[81];
    cout << "Enter the name of a file you wish to open\n";
    cout << "or create: ";
    cin.getline(fileName, 81);
    dataFile.open(fileName, ios::out);
    cout << "The file " << fileName << " was opened.\n";
}
```

Program Output with Example Input

Enter the name of a file you wish to open
or create: **mystuff.dat [Enter]**

The file mystuff.dat was opened.

Table 12-3

File Type	Default Open Mode
ofstream	The file is opened for output only. (Information may be written to the file, but not read from the file.) If the file does not exist, it is created. If the file already exists, its contents are deleted (the file is truncated).
ifstream	The file is opened for input only. (Information may be read from the file, but not written to it.) The file's contents will be read from its beginning. If the file does not exist, the open function fails.

Table 12-4

File Mode Flag	Meaning
ios::app	Append mode. If the file already exists, its contents are preserved and all output is written to the end of the file. By default, this flag causes the file to be created if it does not exist.
ios::ate	If the file already exists, the program goes directly to the end of it. Output may be written anywhere in the file.
ios::binary	Binary mode. When a file is opened in binary mode, information is written to or read from it in pure binary format. (The default mode is text.)
ios::in	Input mode. Information will be read from the file. If the file does not exist, it will not be created and the open function will fail.

Table 12-4 continued

File Mode Flag	Meaning
ios::nocreate	If the file does not already exist, this flag will cause the open function to fail. (The file will not be created.)
ios::noreplace	If the file already exists, this flag will cause the open function to fail. (The existing file will not be opened.)
ios::out	Output mode. Information will be written to the file. By default, the file's contents will be deleted if it already exists.
ios::trunc	If the file already exists, its contents will be deleted (truncated). This is the default mode used by ios::out.

Opening a File at Declaration

```
fstream dataFile("names.dat", ios::in |
   ios::out);
```

Program 12-2

```
// This program demonstrates the opening of a file at the
// time the file stream object is declared.
#include <iostream.h>
#include <fstream.h>

void main(void)
{
    fstream dataFile("names.dat", ios::in | ios::out);
    cout << "The file names.dat was opened.\n";
}
```

Program Output with Example Input

The file names.dat was opened.

Testing for Open Errors

```
dataFile.open("cust.dat", ios::in);
if (!dataFile)
{
   cout << "Error opening file.\n";
}
```

Another way to Test for Open Errors

```
dataFile.open("cust.dat", ios::in);
if (dataFile.fail())
{
   cout << "Error opening file.\n";
}
```

12.6 Closing a File

- A file should be closed when a program is finished using it.

Program 12-3

```
// This program demonstrates the close function.
#include <iostream.h>
#include <fstream.h>
void main(void)
{
    fstream dataFile;
    dataFile.open("testfile.txt", ios::out);
    if (!dataFile)
    {
        cout << "File open error!" << endl;
        return;
    }
    cout << "File was created successfully.\n";
    cout << "Now closing the file.\n";
    dataFile.close();
}
```

Program Output

File was created successfully.
Now closing the file.

12.7 Using << to Write Information to a File

- The stream insertion operator (<<) may be used to write information to a file.

```
outputFile << "I love C++ programming !"
```

Program 12-4

```
// This program uses the << operator to write information to a file.
#include <iostream.h>
#include <fstream.h>

void main(void)
{
    fstream dataFile;
    char line[81];

    dataFile.open("demofile.txt", ios::out);
    if (!dataFile)
    {
        cout << "File open error!" << endl;
        return;
    }
```

Program continues

```
    cout << "File opened successfully.\n";
    cout << "Now writing information to the file.\n";
    dataFile << "Jones\n";
    dataFile << "Smith\n";
    dataFile << "Willis\n";
    dataFile << "Davis\n";
    dataFile.close();
    cout << "Done.\n";
}
```

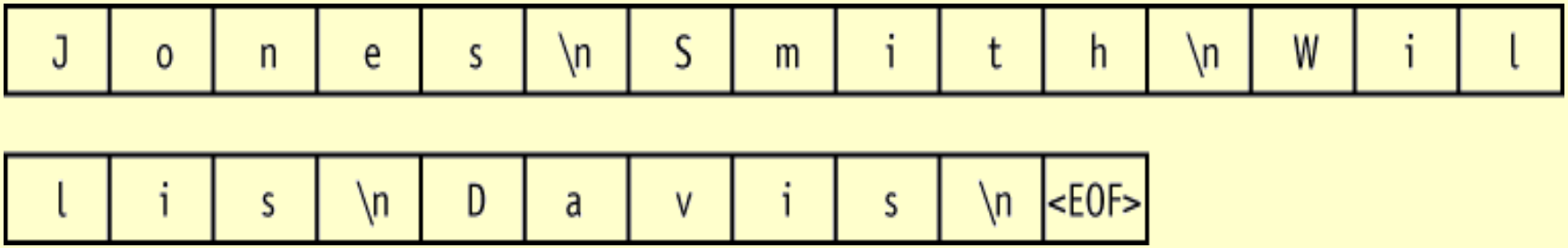
Program Screen Output

File opened successfully.
Now writing information to the file.
Done.

Output to File demofile.txt

Jones
Smith
Willis
Davis

Figure 12-3



Program 12-5

```
// This program writes information to a file, closes the file,
// then reopens it and appends more information.
#include <iostream.h>
#include <fstream.h>
void main(void)
{
    fstream dataFile;

    dataFile.open("demofile.txt", ios::out);
    dataFile << "Jones\n";
    dataFile << "Smith\n";
    dataFile.close();
    dataFile.open("demofile.txt", ios::app);
    dataFile << "Willis\n";
    dataFile << "Davis\n";
    dataFile.close();
}
```

Output to File demofile.txt

Jones
Smith
Willis
Davis

12.8 File Output Formatting

- File output may be formatted the same way as screen output.

Program 12-6

```
// This program uses the precision member function of a
// file stream object to format file output.
#include <iostream.h>
#include <fstream.h>

void main(void)
{
    fstream dataFile;
    float num = 123.456;
    dataFile.open("numfile.txt", ios::out);
    if (!dataFile)
    {
        cout << "File open error!" << endl;
        return;
    }
```

Program continues

```
    dataFile << num << endl;
    dataFile.precision(5);
    dataFile << num << endl;
    dataFile.precision(4);
    dataFile << num << endl;
    dataFile.precision(3);
    dataFile << num << endl;
}
```

Contents of File numfile.txt

123.456
123.46
123.5
124

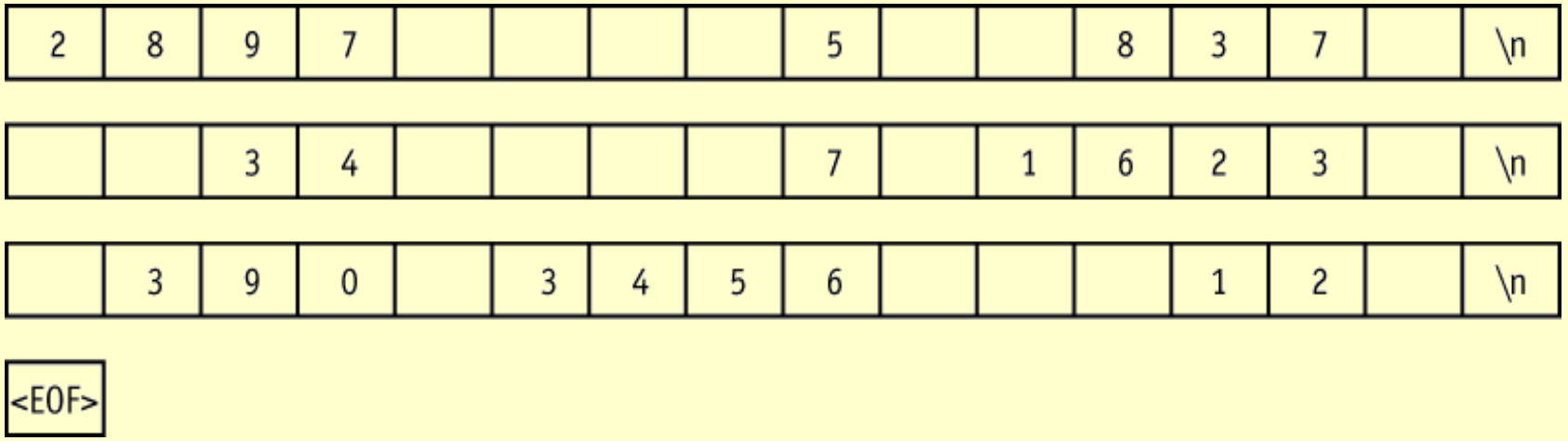
Program 12-7

```
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
void main(void)
{    fstream outFile("table.txt", ios::out);
     int nums[3][3] = { 2897, 5, 837,
                        34, 7, 1623,
                        390, 3456, 12 };
     // Write the three rows of numbers
     for (int row = 0; row < 3; row++)
     {
          for (int col = 0; col < 3; col++)
          {
              outFile << setw(4) << nums[row][col] << "  ";
          }
          outFile << endl;
     }
     outFile.close();
}
```

Contents of File TABLE.TXT

```
2897      5   837
  34      7  1623
 390   3456    12
```

Figure 12-6



12.9 Using >> to Read Information from a File

- The stream extraction operator (>>) may be used to read information from a file.

Program 12-8

```
// This program uses the >> operator to read information from a file.
#include <iostream.h>
#include <fstream.h>

void main(void)
{
    fstream dataFile;
    char name[81];
    dataFile.open("demofile.txt", ios::in);
    if (!dataFile)
    {
        cout << "File open error!" << endl;
        return;
    }
    cout << "File opened successfully.\n";
    cout << "Now reading information from the file.\n\n";
```

Program continues

```
    for (int count = 0; count < 4; count++)
    {
          dataFile >> name;
          cout << name << endl;
    }
    dataFile.close();
    cout << "\nDone.\n";
}
```

Program Screen Output

```
File opened successfully.
Now reading information from the file.

Jones
Smith
Willis
Davis

Done.
```

12.10 Detecting the End of a File

- The `eof()` member function reports when the end of a file has been encountered.

```
if (inFile.eof())
    inFile.close();
```

Program 12-9

```
// This program uses the file stream object's eof() member
// function to detect the end of the file.
#include <iostream.h>
#include <fstream.h>
void main(void)
{
    fstream dataFile;
    char name[81];
    dataFile.open("demofile.txt", ios::in);
    if (!dataFile)
    {
            cout << "File open error!" << endl;
            return;
    }
    cout << "File opened successfully.\n";
    cout << "Now reading information from the file.\n\n";
```

Program continues

```
    dataFile >> name;   // Read first name from the file
    while (!dataFile.eof())
    {
          cout << name << endl;
          dataFile >> name;
    }
    dataFile.close();
    cout << "\nDone.\n";
}
```

Program Screen Output

```
File opened successfully.
Now reading information from the file.

Jones
Smith
Willis
Davis
Done.
```

Note on `eof()`

- In C++, “end of file” doesn’t mean the program is at the last piece of information in the file, but beyond it. The `eof()` function returns true when there is no more information to be read.

12.11 Passing File Stream Objects to Functions

- File stream objects may be passed by reference to functions.

```
bool openFileIn(fstream &file, char name[51])
{
   bool status;

   file.open(name, ios::in);
   if (file.fail())
      status = false;
   else
      status = true;
   return status;
}
```

12.12 More Detailed Error Testing

- All stream objects have error state bits that indicate the condition of the stream.

Table 12-5

Bit	Description
ios::eofbit	Set when the end of an input stream is encountered.
ios::failbit	Set when an attempted operation has failed.
ios::hardfail	Set when an unrecoverable error has occurred.
ios::badbit	Set when an invalid operation has been attempted.
ios::goodbit	Set when all the flags above are not set. Indicates the stream is in good condition.

Table 12-6

Function	Description
eof()	Returns true (non-zero) if the eofbit flag is set, otherwise returns false.
fail()	Returns true (non-zero) if the failbit or hardfail flags are set, otherwise returns false.
bad()	Returns true (non-zero) if the badbit flag is set, otherwise returns false.
good()	Returns true (non-zero) if the goodbit flag is set, otherwise returns false.
clear()	When called with no arguments, clears all the flags listed above. Can also be called with a specific flag as an argument.

Program 12-11

```
// This program demonstrates the return value of the stream
// object error testing member functions.
#include <iostream.h>
#include <fstream.h>
// Function prototype
void showState(fstream &);

void main(void)
{
    fstream testFile("stuff.dat", ios::out);
    if (testFile.fail())
    {
            cout << "cannot open the file.\n";
            return;
    }
```

Program continues

```
int num = 10;
cout << "Writing to the file.\n";
testFile << num;                        // Write the integer to
testFile
showState(testFile);
testFile.close();                       // Close the file
testFile.open("stuff.dat", ios::in);    // Open for input
if (testFile.fail())
{
      cout << "cannot open the file.\n";
      return;
}
```

Program continues

```
    cout << "Reading from the file.\n";
    testFile >> num;                    // Read the only number in the file
    showState(testFile);
    cout << "Forcing a bad read operation.\n";
    testFile >> num;                    // Force an invalid read operation
    showState(testFile);
    testFile.close();                   // Close the file
}

// Definition of function ShowState. This function uses
// an fstream reference as its parameter. The return values of
// the eof(), fail(), bad(), and good() member functions are
// displayed. The clear() function is called before the function
// returns.
```

Program continues

```
void showState(fstream &file)
{
    cout << "File Status:\n";
    cout << "  eof bit: " << file.eof() << endl;
    cout << "  fail bit: " << file.fail() << endl;
    cout << "  bad bit: " << file.bad() << endl;
    cout << "  good bit: " << file.good() << endl;
    file.clear();    // Clear any bad bits
}
```

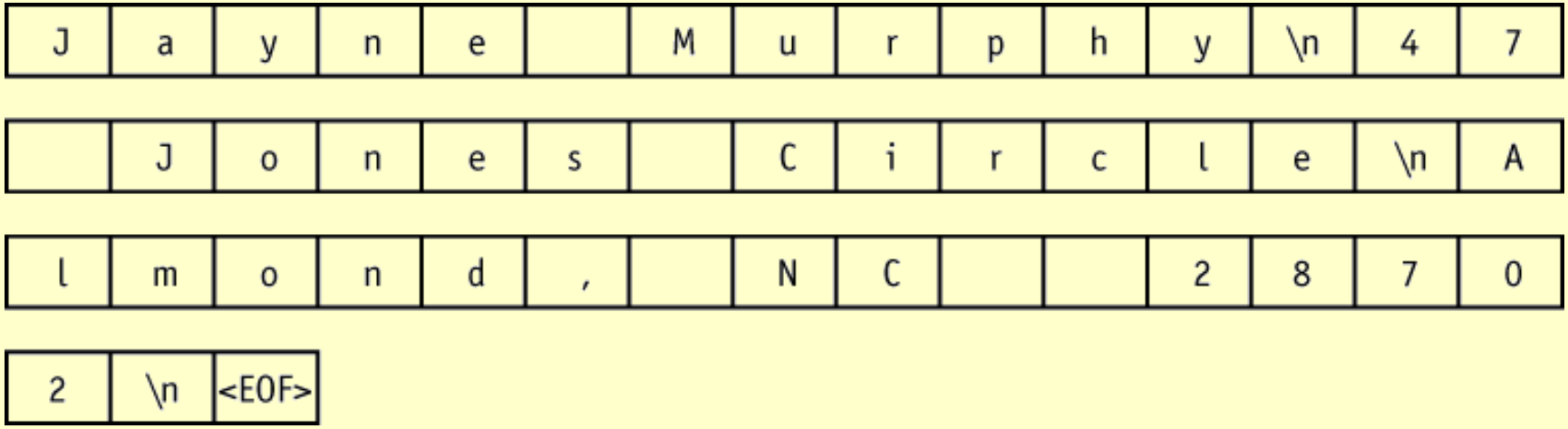
Program Output

```
Writing to the file.
File Status:
  eof bit: 0
  fail bit: 0
  bad bit: 0
  good bit: 1
Reading from the file.
File Status:
  eof bit: 0
  fail bit: 0
  bad bit: 0
  good bit: 1
Forcing a bad read operation.
File Status:
  eof bit: 1
  fail bit: 2
  bad bit: 0
  good bit: 0
```

12.13 Member Functions for Reading and Writing Files

- File stream objects have member functions for more specialized file reading and writing.

Figure 12-8



Program 12-12

```
// This program uses the file stream object's eof() member
// function to detect the end of the file.
#include <iostream.h>
#include <fstream.h>

void main(void)
{
   fstream nameFile;
   char input[81];

   nameFile.open("murphy.txt", ios::in);
   if (!nameFile)
   {
        cout << "File open error!" << endl;
        return;
   }
```

Program 12-12 (continued)

```
   nameFile >> input;
   while (!nameFile.eof())
   {
       cout << input;
       nameFile >> input;
   }
   nameFile.close();
}
```

Program Screen Output

JayneMurphy47JonesCircleAlmond,NC28702

The getline Member Function

- dataFile.getline(str, 81, '\n');

  str – This is the name of a character array, or a pointer to a section of memory. The information read from the file will be stored here.

  81 – This number is one greater than the maximum number of characters to be read. In this example, a maximum of 80 characters will be read.

  '\n' – This is a delimiter character of your choice. If this delimiter is encountered, it will cause the function to stop reading before it has read the maximum number of characters. (This argument is optional. If it's left our, '\n' is the default.)

Program 12-13

```
// This program uses the file stream object's getline member
// function to read a line of information from the file.

#include <iostream.h>
#include <fstream.h>

void main(void)
{
   fstream nameFile;
   char input[81];

   nameFile.open("murphy.txt", ios::in);
   if (!nameFile)
   {
        cout << "File open error!" << endl;
        return;
   }
```

Program continues

```
   nameFile.getline(input, 81);  // use \n as a delimiter
   while (!nameFile.eof())
   {
        cout << input << endl;
        nameFile.getline(input, 81); // use \n as a delimiter
   }
   nameFile.close();
}
```

Program Screen Output

Jayne Murphy
47 Jones Circle
Almond, NC 28702

Program 12-14

```
// This file shows the getline function with a user-
// specified delimiter.

#include <iostream.h>
#include <fstream.h>
void main(void)
{
   fstream dataFile("names2.txt", ios::in);
   char input[81];

   dataFile.getline(input, 81, '$');
   while (!dataFile.eof())
   {
        cout << input << endl;
        dataFile.getline(input, 81, '$');
   }
   dataFile.close();
}
```

Program Output

```
Jayne Murphy
47 Jones Circle
Almond, NC 28702

Bobbie Smith
217 Halifax Drive
Canton, NC 28716

Bill Hammet
PO Box 121
Springfield, NC 28357
```

The `get` Member Function

```
inFile.get(ch);
```

Program 12-15

```
// This program asks the user for a file name. The file is
// opened and its contents are displayed on the screen.
#include <iostream.h>
#include <fstream.h>

void main(void)
{
   fstream file;
   char ch, fileName[51];
   cout << "Enter a file name: ";
   cin >> fileName;
   file.open(fileName, ios::in);
   if (!file)
   {
        cout << fileName << " could not be opened.\n";
        return;
   }
```

Program continues

```
   file.get(ch);   // Get a character
   while (!file.eof())
   {
       cout << ch;
       file.get(ch);   // Get another character
   }
   file.close();
}
```

The `put` Member Function

- `outFile.put(ch);`

Program 12-16

```
// This program demonstrates the put member function.
#include <iostream.h>
#include <fstream.h>

void main(void)
{  fstream dataFile("sentence.txt", ios::out);
   char ch;

   cout << "Type a sentence and be sure to end it with a ";
   cout << "period.\n";
   while (1)
   {
        cin.get(ch);
        dataFile.put(ch);
        if (ch == '.')
             break;
   }
   dataFile.close();
}
```

Program Screen Output with Example Input

```
Type a sentence and be sure to end it with a
period.
I am on my way to becoming a great programmer. [Enter]
```

Resulting Contents of the File SENTENCE.TXT*:*

```
I am on my way to becoming a great programmer.
```

12.14 Focus on Software Engineering: Working with Multiple Files

- It's possible to have more than one file open at once in a program.

Program 12-17

```
// This program demonstrates reading from one file and writing
// to a second file.
#include <iostream.h>
#include <fstream.h>
#include <ctype.h> // Needed for the toupper function

void main(void)
{
   ifstream inFile;
   ofstream outFile("out.txt");
   char fileName[81], ch, ch2;

   cout << "Enter a file name: ";
   cin >> fileName;
   inFile.open(fileName);
   if (!inFile)
   {
       cout << "Cannot open " << fileName << endl;
       return;
   }
```

Program continues

```
   inFile.get(ch);          // Get a characer from file 1
   while (!inFile.eof())    // Test for end of file
   {
       ch2 = toupper(ch); // Convert to uppercase
       outFile.put(ch2);  // Write to file2
        inFile.get(ch);     // Get another character from file 1
   }
   inFile.close();
   outFile.close();
   cout << "File conversion done.\n";
}
```

Program Screen Output with Example Input

```
Enter a file name: hownow.txt [Enter]
File conversion done.
```

Contents of hownow.txt:

```
how now brown cow.
How Now?
```

Resulting Contents of out.txt:

```
HOW NOW BROWN COW.
HOW NOW?
```

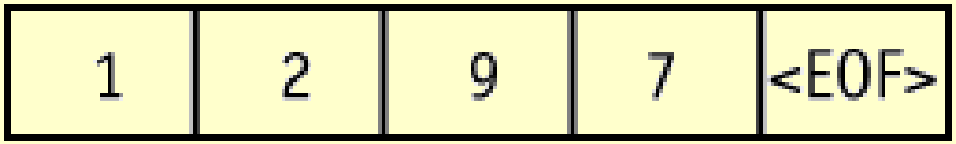
12.15 Binary Files

- Binary files contain data that is unformatted, and not necessarily stored as ASCII text.

```
file.open("stuff.dat", ios::out | ios::binary);
```

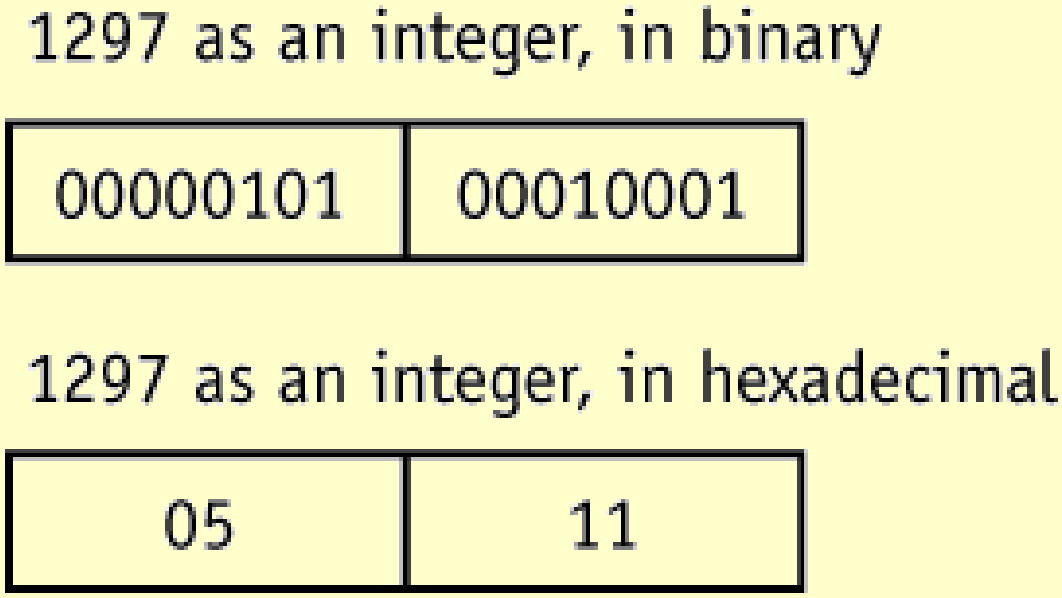
Figure 12-9

1297 expressed in Character format



1297 expressed in ASCII

Figure 12-10


1297 as an integer, in binary
00000101
00010001
1297 as an integer, in hexadecimal
05
11

Program 12-18

```
// This program uses the write and read functions.
#include <iostream.h>
#include <fstream.h>

void main(void)
{
   fstream file("NUMS.DAT", ios::out | ios::binary);
   int buffer[10] = {1,  2,  3,  4,  5,  6,  7,  8,  9,  10};

   cout << "Now writing the data to the file.\n";
   file.write((char*)buffer, sizeof(buffer));
   file.close();
   file.open("NUMS.DAT", ios::in);  // Reopen the file.
   cout << "Now reading the data back into memory.\n";
   file.read((char*)buffer, sizeof(buffer));
   for (int count = 0; count < 10; count++)
       cout << buffer[count] << " ";
   file.close();
}
```

Program Screen Output

```
Now writing the data to the file.
Now reading the data back into memory.
1 2 3 4 5 6 7 8 9 10
```

12.16 Creating Records with Structures

- Structures may be used to store fixed-length records to a file.

```
struct Info
{
   char name[51];
   int age;
   char address1[51];
   char address2[51];
   char phone[14];
};
```

- Since structures can contain a mixture of data types, you should always use the ios::binary mode when opening a file to store them.

Program 12-19

```
// This program demonstrates the use of a structure variable to
// store a record of information to a file.

#include <iostream.h>
#include <fstream.h>
#include <ctype.h>  // for toupper

// Declare a structure for the record.
struct Info
{
   char name[51];
   int age;
   char address1[51];
   char address2[51];
   char phone[14];
};
```

Program continues

```
void main(void)
{
   fstream people("people.dat", ios::out | ios::binary);
   Info person;
   char again;
   if (!people)
   {
       cout << "Error opening file. Program aborting.\n";
       return;
   }
   do
   {
       cout << "Enter the following information about a "
            << "person:\n";
       cout << "Name: ";
```

Program continues

```
        cin.getline(person.name, 51);
        cout << "Age: ";
        cin >> person.age;
        cout << "Address line 1: ";
        cin.getline(person.address1, 51);
        cout << "Address line 2: ";
        cin.getline(person.address2, 51);
        cout << "Phone: ";
        cin.getline(person.phone, 14);
        people.write((char *)&person, sizeof(person));
        cout << "Do you want to enter another record? ";
        cin >> again;
        cin.ignore();
    } while (toupper(again) == 'Y');
    people.close();
}
```

Program Screen Output with Example Input

```
Enter the following information about a person:
Name: Charlie Baxter [Enter]
Age: 42 [Enter]
Address line 1: 67 Kennedy Bvd. [Enter]
Address line 2: Perth, SC  38754 [Enter]
Phone: (803)555-1234 [Enter]
Do you want to enter another record? Y [Enter]
Enter the following information about a person:
Name: Merideth Murney [Enter]
Age: 22 [Enter]
Address line 1: 487 Lindsay Lane [Enter]
Address line 2: Hazelwood, NC  28737 [Enter]
Phone: (704)453-9999 [Enter]
Do you want to enter another record? N [Enter]
```

12.17 Random Access Files

- Random Access means non-sequentially accessing informaiton in a file.

Figure 12-11

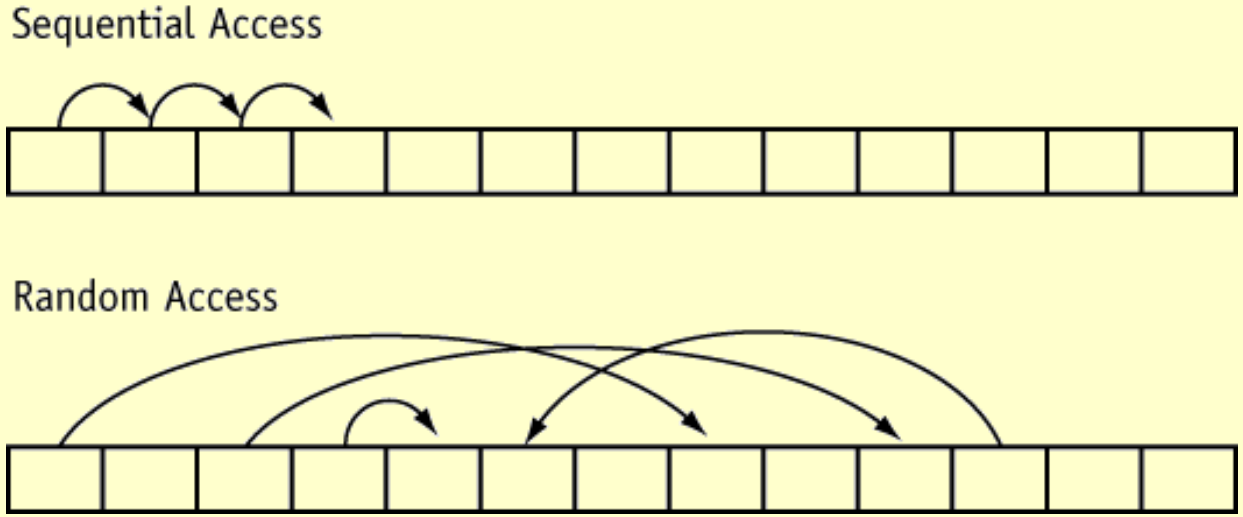


Table 12-7

Mode Flag	Description
ios::beg	The offset is calculated from the beginning of the file.
ios::end	The offset is calculated from the end of the file.
ios::cur	The offset is calculated from the current position.

Table 12-8

Statement	How it Affects the Read/Write Position
File.seekp(32L, ios::beg);	Sets the write position to the 33rd byte (byte 32) from the beginning of the file.
file.seekp(-10L, ios::end);	Sets the write position to the 11th byte (byte 10) from the end of the file.
file.seekp(120L, ios::cur);	Sets the write position to the 121st byte (byte 120) from the current position.
file.seekg(2L, ios::beg);	Sets the read position to the 3rd byte (byte 2) from the beginning of the file.
file.seekg(-100L, ios::end);	Sets the read position to the 101st byte (byte 100) from the end of the file.
file.seekg(40L, ios::cur);	Sets the read position to the 41st byte (byte 40) from the current position.
file.seekg(0L, ios::end);	Sets the read position to the end of the file.

Program 12-21

```
// This program demonstrates the seekg function.

#include <iostream.h>
#include <fstream.h>

void main(void)
{
   fstream file("letters.txt", ios::in);
   char ch;

   file.seekg(5L, ios::beg);
   file.get(ch);
   cout << "Byte 5 from beginning: " << ch << endl;
   file.seekg(-10L, ios::end);
   file.get(ch);
   cout << "Byte 10 from end: " << ch << endl;
```

Program continues

```
   file.seekg(3L, ios::cur);
   file.get(ch);
   cout << "Byte 3 from current: " << ch << endl;
   file.close();
}
```

Program Screen Output

```
Byte 5 from beginning: f
Byte 10 from end: q
Byte 3 from current: u
```

The `tellp` and `tellg` Member Functions

- `tellp` returns a long integer that is the current byte number of the file's write position.
- `tellg` returns a long integer that is the current byte number of the file's read position.

Program 12-23

```
// This program demonstrates the tellg function.
#include <iostream.h>
#include <fstream.h>
#include <ctype.h>   // For toupper

void main(void)
{
   fstream file("letters.txt", ios::in);
   long offset;
   char ch, again;

   do
   {
        cout << "Currently at position " << file.tellg() << endl;
        cout << "Enter an offset from the beginning of the file: ";
        cin >> offset;
```

Program continues

```
        file.seekg(offset, ios::beg);
        file.get(ch);
        cout << "Character read: " << ch << endl;
        cout << "Do it again? ";
        cin >> again;
    } while (toupper(again) == 'Y');
    file.close();
}
```

Program Output with Example Input

```
Currently at position 0
Enter an offset from the beginning of the file: 5 [Enter]
Character read: f
Do it again? y [Enter]
Currently at position 6
Enter an offset from the beginning of the file: 0 [Enter]
Character read: a
Do it again? y [Enter]
Currently at position 1
Enter an offset from the beginning of the file: 20 [Enter]
Character read: u
Do it again? n [Enter]
```

12.18 Opening a File for Both Input and Output

- You may perform input and output on an fstream file without closing it and reopening it.

```
fstream file("data.dat", ios::in | ios::out);
```

Program 12-24

```
// This program sets up a file of blank inventory records.
#include <iostream.h>
#include <fstream.h>

// Declaration of Invtry structure
struct Invtry
{
   char desc[31];
   int qty;
   float price;
};

void main(void)
{
   fstream inventory("invtry.dat", ios::out | ios::binary);
   Invtry record = { "", 0, 0.0 };
```

Program continues

```
   // Now write the blank records
   for (int count = 0; count < 5; count++)
   {
        cout << "Now writing record " << count << endl;
        inventory.write((char *)&record, sizeof(record));
   }
   inventory.close();
}
```

Program Screen Output

```
Now writing record 0
Now writing record 1
Now writing record 2
Now writing record 3
Now writing record 4
```

Program 12-25

```
// This program displays the contents of the inventory file.
#include <iostream.h>
#include <fstream.h>

// Declaration of Inventory structure
struct Invtry
{
   char desc[31];
   int qty;
   float price;
};

void main(void)
{
   fstream inventory("invtry.dat", ios::in | ios::binary);
   Invtry record = { "", 0, 0.0 };
```

Program continues

```
   // Now read and display the records
   inventory.read((char *)&record, sizeof(record));
   while (!inventory.eof())
   {
       cout << "Description: ";
       cout << record.desc << endl;
       cout << "Quantity: ";
       cout << record.qty << endl;
       cout << "Price: ";
       cout << record.price << endl << endl;
       inventory.read((char *)&record, sizeof(record));
   }
   inventory.close();
}
```

Here is the screen output of Program 12-25 if it is run immediately after Program 12-24 sets up the file of blank records.

Program Screen Output

```
Description:
Quantity: 0
Price: 0.0
Description:
Quantity: 0
Price: 0.0
Description:
Quantity: 0
Price: 0.0
Description:
Quantity: 0
Price: 0.0
Description:
Quantity: 0
Price: 0.0
Description:
Quantity: 0
Price: 0.0
```

Program 12-26

```
// This program allows the user to edit a specific record in
// the inventory file.
#include <iostream.h>
#include <fstream.h>

// Declaration of Invtry structure
struct Invtry
{
   char desc[31];
   int qty;
   float price;
};

void main(void)
{
```

Program continues

```
fstream inventory("invtry.dat", ios::in | ios::out |  ios::binary);
Invtry record;
long recNum;
cout << "Which record do you want to edit?";
cin >> recNum;
inventory.seekg(recNum * sizeof(record), ios::beg);
inventory.read((char *)&record, sizeof(record));
cout << "Description: ";
cout << record.desc << endl;
cout << "Quantity: ";
cout << record.qty << endl;
cout << "Price: ";
cout << record.price << endl;
cout << "Enter the new data:\n";
cout << "Description: ";
```

Program continues

```
   cin.getline(record.desc, 31);
   cout << "Quantity: ";
   cin >> record.qty;
   cout << "Price: ";
   cin >> record.price;
   inventory.seekp(recNum * sizeof(record), ios::beg);
   inventory.write((char *)&record, sizeof(record));
   inventory.close();
}
```

Program Screen Output with Example Input

```
Which record do you ant to edit? 2 [Enter]
Description:
Quantity: 0
Price: 0.0
Enter the new data:
Description: Wrench [Enter]
Quantity: 10 [Enter]
Price: 4.67 [Enter]
```