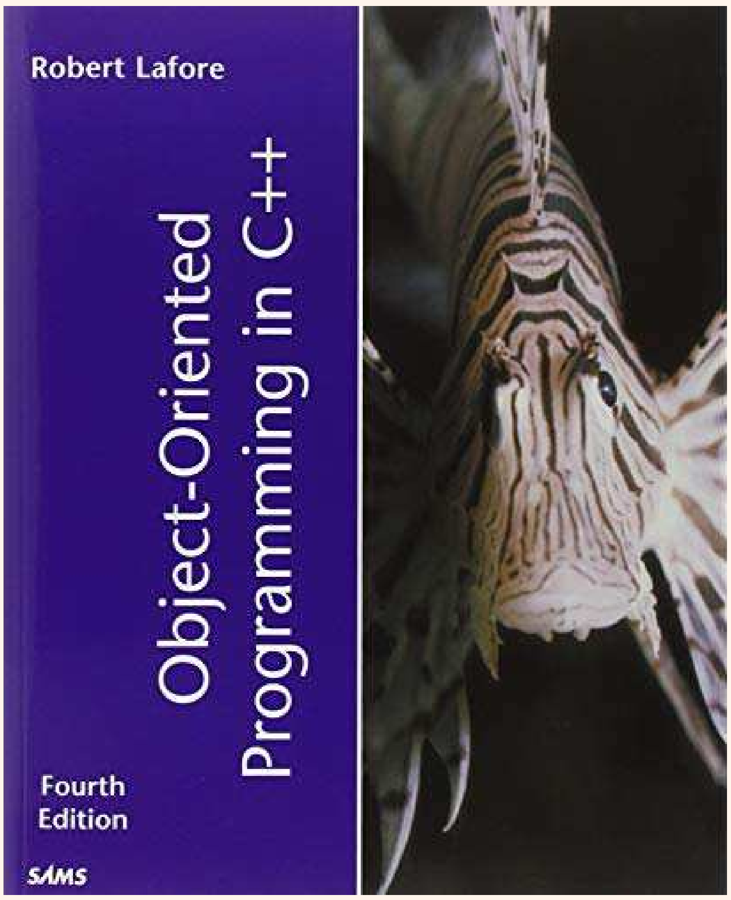


Chapter 12

Streams and Files

Animated Version

Topics

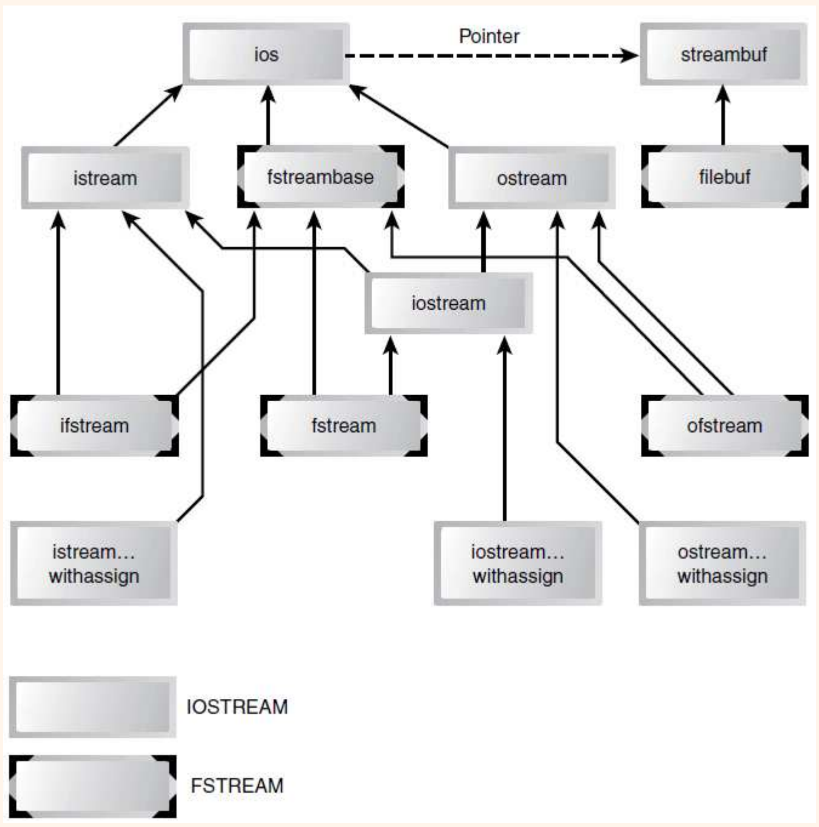
- Stream Classes
- Stream Errors
- Disk File I/O with Streams
- File Pointers
- Error Handling in File I/O
- File I/O with Member Functions

Stream Classes

- A stream is a general name given to a flow of data.
- In C++ a stream is represented by an object of a particular class.
  - Example: cin and cout, ifstream
- Advantages of Streams
  - Simplicity: not much use of formatting characters (%d, %f)
  - can overload existing operators and functions, such as the insertion (<<) and extraction (>>) operators, to work with classes that you create. This makes your own classes work in the same way as the built-in types, which again makes programming easier and more error free.

Stream Classes (2)

- The Stream Class Hierarchy



The `ios` Class

- Formatting Flags

```
cout.setf(ios::left);    // left justify output text
cout >> "This text is left-justified";
cout.unsetf(ios::left);  // return to default (right justified)
```

TABLE 12.1 ios Formatting Flags

Flag	*Meaning*
`skipws`	Skip (ignore) whitespace on input
`left`	Left-adjust output [`12.34       `]
`right`	Right-adjust output [`       12.34`]
`internal`	Use padding between sign or base indicator and number [`+       12.34`]
`dec`	Convert to decimal
`oct`	Convert to octal
`hex`	Convert to hexadecimal
`boolalpha`	Convert bool to "true" or "false" strings
`showbase`	Use base indicator on output (0 for octal, 0x for hex)
`showpoint`	Show decimal point on output
`uppercase`	Use uppercase `X`, `E`, and hex output letters (`ABCDEF`)—the default is lowercase
`showpos`	Display + before positive integers
`scientific`	Use exponential format on floating-point output [`9.1234E2`]
`fixed`	Use fixed format on floating-point output [`912.34`]
`unitbuf`	Flush all streams after insertion
`stdio`	Flush `stdout`, `stderror` after insertion

The `ios` Class

- Manipulators

```
cout << "To each his own." << endl;
```

```
cout << setiosflags(ios::fixed)      // use fixed decimal point
     << setiosflags(ios::showpoint)  // always show decimal point
     << var;
```

```
cout << hex << var;
```

TABLE 12.2 No-Argument ios Manipulators

Manipulator	*Purpose*
`ws`	Turn on whitespace skipping on input
`dec`	Convert to decimal
`oct`	Convert to octal
`hex`	Convert to hexadecimal
`endl`	Insert newline and flush the output stream
`ends`	Insert null character to terminate an output string
`flush`	Flush the output stream
`lock`	Lock file handle
`unlock`	Unlock file handle

TABLE 12.3 ios Manipulators with Arguments

Manipulator	*Argument*	*Purpose*
`setw()`	field width (`int`)	Set field width for output
`setfill()`	fill character (`int`)	Set fill character for output (default is a space)
`setprecision()`	precision (`int`)	Set precision (number of digits displayed)
`setiosflags()`	formatting flags (`long`)	Set specified flags
`resetiosflags()`	formatting flags (`long`)	Clear specified flags

The `ios` Class

- Functions

TABLE 12.4 ios Functions

Function	*Purpose*
`ch = fill();`	Return the fill character (fills unused part of field; default is space)
`fill(ch);`	Set the fill character
`p = precision();`	Get the precision (number of digits displayed for floating-point)
`precision(p);`	Set the precision
`w = width();`	Get the current field width (in characters)
`width(w);`	Set the current field width
`setf(flags);`	Set specified formatting flags (for example, `ios::left`)
`unsetf(flags);`	Unset specified formatting flags
`setf(flags, field);`	First clear field, then set flags

TABLE 12.5 Two-Argument Version of `setf()`

First Argument: Flags to Set	*Second Argument: Field to Clear*
`dec`, `oct`, `hex`	`basefield`
`left`, `right`, `internal`	`adjustfield`
`scientific`, `fixed`	`floatfield`

```
cout.width(14);
cout.fill('*');
cout.setf(ios::left);
cout.unsetf(ios::left);
cout.setf(ios::left, ios::adjustfield);
```

The `istream` Class

- Derived from `ios`.

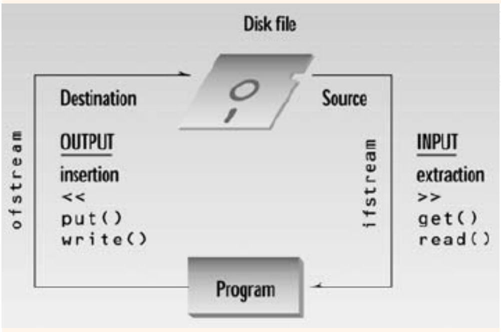


TABLE 12.6 istream Functions

Function	*Purpose*
`>>`	Formatted extraction for all basic (and overloaded) types.
`get(ch);`	Extract one character into `ch`.
`get(str)`	Extract characters into array `str`, until '`\n`'.
`get(str, MAX)`	Extract up to `MAX` characters into array.
`get(str, DELIM)`	Extract characters into array `str` until specified delimiter (typically '`\n`'). Leave delimiting char in stream.
`get(str, MAX, DELIM)`	Extract characters into array `str` until `MAX` characters or the `DELIM` character. Leave delimiting char in stream.
`getline(str, MAX, DELIM)`	Extract characters into array `str`, until `MAX` characters or the `DELIM` character. Extract delimiting character.
`putback(ch)`	Insert last character read back into input stream.
`ignore(MAX, DELIM)`	Extract and discard up to `MAX` characters until (and including) the specified delimiter (typically '`\n`').
`peek(ch)`	Read one character, leave it in stream.
`count = gcount()`	Return number of characters read by a (immediately preceding) call to `get()`, `getline()`, or `read()`.
`read(str, MAX)`	For files—extract up to `MAX` characters into `str`, until `EOF`.
`seekg()`	Set distance (in bytes) of file pointer from start of file.
`seekg(pos, seek_dir)`	Set distance (in bytes) of file pointer from specified place in file. `seek_dir` can be `ios::beg`, `ios::cur`, `ios::end`.
`pos = tellg(pos)`	Return position (in bytes) of file pointer from start of file.

The `ostream` Class

- Derived from `ios`.

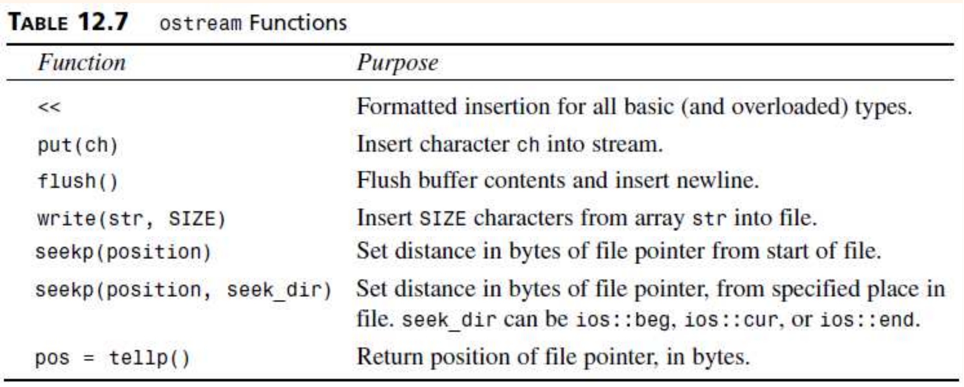


TABLE 12.7 ostream Functions

Function	*Purpose*
`<<`	Formatted insertion for all basic (and overloaded) types.
`put(ch)`	Insert character `ch` into stream.
`flush()`	Flush buffer contents and insert newline.
`write(str, SIZE)`	Insert `SIZE` characters from array `str` into file.
`seekp(position)`	Set distance in bytes of file pointer from start of file.
`seekp(position, seek_dir)`	Set distance in bytes of file pointer, from specified place in file. `seek_dir` can be `ios::beg`, `ios::cur`, or `ios::end`.
`pos = tellp()`	Return position of file pointer, in bytes.

Stream Errors

- Error-Status Bits

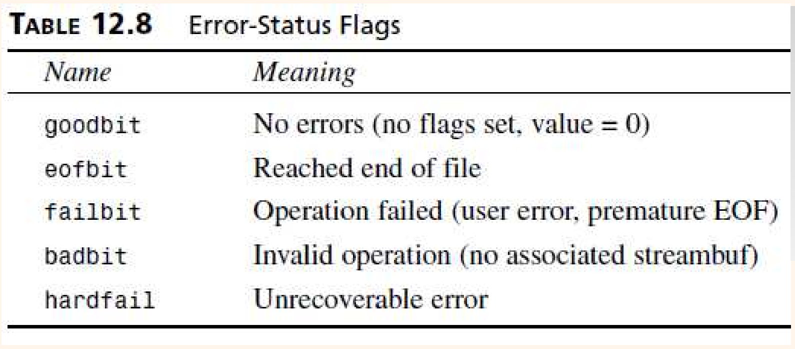
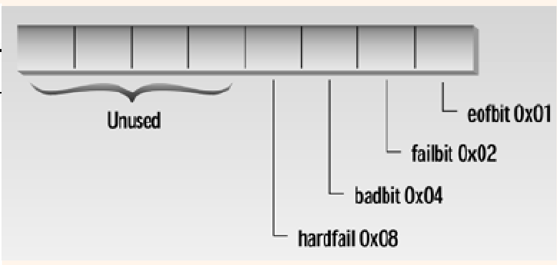


TABLE 12.8 Error-Status Flags

Name	*Meaning*
`goodbit`	No errors (no flags set, value = 0)
`eofbit`	Reached end of file
`failbit`	Operation failed (user error, premature EOF)
`badbit`	Invalid operation (no associated streambuf)
`hardfail`	Unrecoverable error



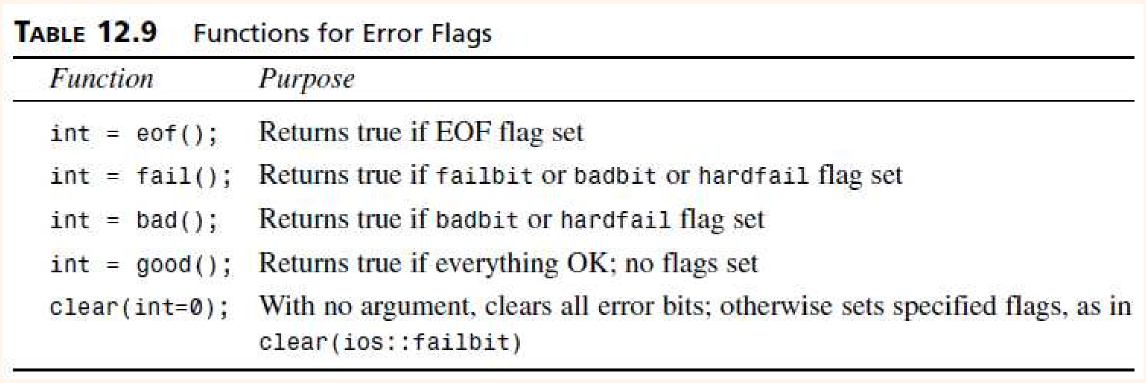


TABLE 12.9 Functions for Error Flags

Function	*Purpose*
`int = eof();`	Returns true if EOF flag set
`int = fail();`	Returns true if `failbit` or `badbit` or `hardfail` flag set
`int = bad();`	Returns true if `badbit` or `hardfail` flag set
`int = good();`	Returns true if everything OK; no flags set
`clear(int=0);`	With no argument, clears all error bits; otherwise sets specified flags, as in `clear(ios::failbit)`

Stream Errors (2)

- Inputting numbers

```
while(true)                               // cycle until input OK
   {
   cout << "\nEnter an integer: ";
   cin >> i;
   if( cin.good() )                    // if no errors
      {
      cin.ignore(10, '\n');            // remove newline
      break;                           // exit loop
      }
   cin.clear();                        // clear the error bits
   cout << "Incorrect input";
   cin.ignore(10, '\n');               // remove newline
   }
cout << "integer is " << i;            // error-free integer
```

- Too many characters
- No-Input Input

Inputting Strings and Characters

- Error-Free Distances

```
// englerr.cpp
// input checking with English Distance class
#include <iostream>
#include <string>
#include <cstdlib>                  //for atoi(), atof()
using namespace std;
int isFeet(string);                 //declaration
////////////////////////////////////////////////////
class Distance                //English Distance class
   {
   private:
      int feet;
      float inches;
   public:
      Distance()                //constructor (no args)
         { feet = 0; inches = 0.0; }
      Distance(int ft, float in)//constructor (two args)
         { feet = ft; inches = in; }
      void showdist()              //display distance
         { cout << feet << "\'-" << inches << '\"'; }
      void getdist();              //get length from user
   };
//----------------------------------------------------
void Distance::getdist()         //get length from user
   {
   string instr;                    //for input string

   while(true)               //cycle until feet are right
      {
      cout << "\n\nEnter feet: ";
      cin.unsetf(ios::skipws);//do not skip white space
      cin >> instr;                //get feet as a string
```

```
 if( isFeet(instr) )  //is it a correct feet value?
        {                       //yes
   cin.ignore(10, '\n');//eat chars, including newline
   feet = atoi( instr.c_str() );//convert to integer
        break;                  //break out of 'while'
        }                       //no, not an integer
    cin.ignore(10, '\n');//eat chars, including newline
    cout << "Feet must be an integer less than 1000\n";
      }  //end while feet

   while(true)           //cycle until inches are right
      {
      cout << "Enter inches: ";
      cin.unsetf(ios::skipws); //do not skip white space
      cin >> inches;         //get inches (type float)
      if(inches>=12.0 || inches<0.0)
         {
         cout << "Inches must be between 0.0 and
11.99\n";
         cin.clear(ios::failbit);

           //"artificially" set fail bit
         }
      if( cin.good() )     //check for cin failure
         {                 //(most commonly a non-digit)
         cin.ignore(10, '\n');    //eat the newline
         break;            //input is OK, exit 'while'
         }
      cin.clear();        //error; clear the error state
    cin.ignore(10, '\n');//eat chars, including newline
    cout << "Incorrect inches input\n";  //start again
      }  //end while inches
   }
//----------------------------------------------------
```

Inputting Strings and Characters

- Error-Free Distances (2)

```
int isFeet(string str)      //return true if the string
   {                        //   is a correct feet value
   int slen = str.size();         //get length
   if(slen==0 || slen > 5)   //if no input, or too long
      return 0;                   //not an int
   for(int j=0; j<slen; j++)      //check each character
                             //if not digit or minus
      if( (str[j] < '0' || str[j] > '9') && str[j] != '-
  ' )
         return 0;         //string is not correct feet
   double n = atof( str.c_str() );  //convert to double
   if( n<-999.0 || n>999.0 )      //is it out of range?
      return 0;                 //if so, not correct feet
   return 1;                      //it is correct feet
   }
/////////////////////////////////////////////////////////
int main()
   {
   Distance d;               //make a Distance object
   char ans;
   do
      {
      d.getdist();           //get its value from user
      cout << "\nDistance = ";
      d.showdist();               //display it
      cout << "\nDo another (y/n)? ";
      cin >> ans;
     cin.ignore(10, '\n');//eat chars, including newline
      } while(ans != 'n');        //cycle until 'n'
   return 0;
   }
```

Disk File I/O with Streams

- Formatted File I/O (Writing Data)

```
// formato.cpp
// writes formatted output to a file, using <<
#include <fstream>                 //for file I/O
#include <iostream>
#include <string>
using namespace std;
int main()
   {
   char ch = 'x';
   int j = 77;
   double d = 6.02;
   string str1 = "Kafka";         //strings without
   string str2 = "Proust";     //   embedded spaces

   ofstream outfile("fdata.txt");
                          //create ofstream object
   outfile << ch             //insert (write) data
           << j
           << ' '      //needs space between numbers
           << d
           << str1
           << ' '     //needs spaces between strings
           << str2;
   cout << "File written\n";
   return 0;
   }
```

- Formatted File I/O (Reading Data)

```
// formati.cpp
// reads formatted output from a file, using >>
#include <fstream>              //for file I/O
#include <iostream>
#include <string>
using namespace std;
int main()
   {
   char ch;
   int j;
   double d;
   string str1;
   string str2;

   ifstream infile("fdata.txt");
                     //create ifstream object
                     //extract (read) data from it
   infile >> ch >> j >> d >> str1 >> str2;

   cout << ch << endl        //display the data
        << j << endl
        << d << endl
        << str1 << endl
        << str2 << endl;
   return 0;
   }
```

```
x
77
6.02
Kafka
Proust
```

- `ifstream` for i/p, `fstream` for both i/p & o/p, & `ofstream` for o/p.
- C++ approach is considerably cleaner and easier to implement.

Disk File I/O with Streams (2)

- Character I/O: Writing and Reading

```
// oline.cpp
// file output with strings
#include <fstream>            //for file I/O
using namespace std;

int main()
   {
   ofstream outfile("TEST.TXT");  //create file
    for output
                     //send text to file
   outfile << "I fear thee, ancient Mariner!\n";
   outfile << "I fear thy skinny hand\n";
   outfile << "And thou art long, and lank, and
    brown,\n";
   outfile << "As is the ribbed sea sand.\n";
   return 0;
   }
```

```
// iline.cpp
// file input with strings
#include <fstream>                  //for file
    functions
#include <iostream>
using namespace std;

int main()
   {
   const int MAX = 80;       //size of buffer
   char buffer[MAX];          //character buffer
   ifstream infile("TEST.TXT");
                         //create file for input
   while( !infile.eof() ) //until end-of-file
   //while( infile.good() )
   //while( infile )
      {
      infile.getline(buffer, MAX);
                         //read a line of text
      cout << buffer << endl;   //display it
      }
   return 0;
   }
```

Disk File I/O with Streams (3)

- Character I/O: Writing and Reading

```
// ochar.cpp
// file output with characters
#include <fstream>          //for file functions
#include <iostream>
#include <string>
using namespace std;

int main()
   {
   string str = "Time is a great teacher, but
    unfortunately "
             "it kills all its pupils.
    Berlioz";

   ofstream outfile("TEST.TXT");
                    //create file for output
                    //for each character,
   for(int j=0; j<str.size(); j++)
        outfile.put( str[j] );//write it to file
   cout << "File written\n";
   return 0;
   }
```

```
// ichar.cpp
// file input with characters
#include <fstream>                //for file
    functions
#include <iostream>
using namespace std;

int main()
   {
   char ch;               //character to read
   ifstream infile("TEST.TXT");
                        //create file for input
   while( infile )    //read until EOF or error
      {
      infile.get(ch);    //read character
      cout << ch;         //display it
      }
   cout << endl;
   return 0;
   }
```

```
int main()
   {
   ifstream infile("TEST.TXT");

   cout << infile.rdbuf();
   cout << endl;
   return 0;
   }
```

Disk File I/O with Streams (4)

- Binary I/O

```
// binio.cpp
// binary input and output with integers
#include <fstream>                      //for file streams
#include <iostream>
using namespace std;
const int MAX = 100;                    //size of buffer
int buff[MAX];                          //buffer for integers

int main()
   {
   for(int j=0; j<MAX; j++)             //fill buffer with data
      buff[j] = j;                      //(0, 1, 2, ...)
                                        //create output stream
   ofstream os("edata.dat", ios::binary);
                                        //write to it
   os.write( reinterpret_cast<char*>(buff), MAX*sizeof(int) );
   os.close();                          //must close it

   for(j=0; j<MAX; j++)                 //erase buffer
      buff[j] = 0;
                                        //create input stream
   ifstream is("edata.dat", ios::binary);      //read from it
   is.read( reinterpret_cast<char*>(buff), MAX*sizeof(int) );

   for(j=0; j<MAX; j++)                 //check data
      if( buff[j] != j )
         { cerr << "Data is incorrect\n"; return 1; }
   cout << "Data is correct\n";
   return 0;
   }
```

- The reinterpret_cast Operator:
  - Used here to make it possible for a buffer of type int to look to the read() and write() functions like a buffer of type char.

Disk File I/O with Streams (5)

- Object I/O: Writing and Reading

```
// opers.cpp
// saves person object to disk
#include <fstream>            //for file streams
#include <iostream>
using namespace std;
///////////////////////////////////////////////
class person            //class of persons
   {
   protected:
      char name[80];          //person's name
      short age;              //person's age
   public:
      void getData()        //get person's data
         {
         cout << "Enter name: "; cin >> name;
         cout << "Enter age: "; cin >> age;
         }
   };
//////////////////////////////////////////////
int main()
   {
   person pers;           //create a person
   pers.getData();      //get data for person
                      //create ofstream object
   ofstream outfile("PERSON.DAT", ios::binary);
                                   //write to it
   outfile.write(reinterpret_cast<char*>(&pers),
    sizeof(pers));
   return 0;
   }
```

```
Enter name: Coleridge
Enter age: 62
```

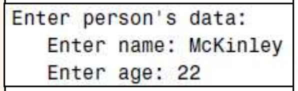
```
// ipers.cpp
// reads person object from disk
#include <fstream>            //for file streams
#include <iostream>
using namespace std;
///////////////////////////////////////////////
class person            //class of persons
   {
   protected:
      char name[80];      //person's name
      short age;          //person's age
   public:
      void showData()   //display person's data
         {
         cout << "Name: " << name << endl;
         cout << "Age: " << age << endl;
         }
   };
//////////////////////////////
int main()
   {
   person pers;        //create person variable
   ifstream infile("PERSON.DAT", ios::binary);
    //create stream
    //read stream
   infile.read( reinterpret_cast<char*>(&pers),
    sizeof(pers) );
   pers.showData();        //display person
   return 0;
   }
```

```
Name: Coleridge
Age: 62
```

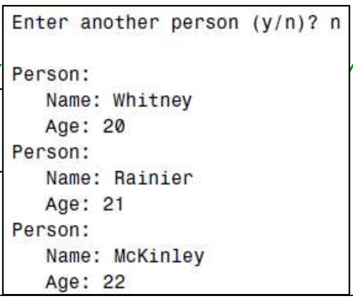
Disk File I/O with Streams (6)

- I/O with Multiple Objects

```cpp
// diskfun.cpp
// reads and writes several objects to disk
#include <fstream>          //for file streams
#include <iostream>
using namespace std;
///////////////////////////////////////////////
class person            //class of persons
{  protected:
      char name[80];    //person's name
      int age;          //person's age
   public:
      void getData()    //get person's data
         {
       cout << "\n   Enter name: "; cin >> name;
       cout << "   Enter age: "; cin >> age;
         }
      void showData()   //display person's data
         {
         cout << "\n   Name: " << name;
         cout << "\n   Age: " << age;
         }
   };
///////////////////                            //
```



```
Enter person's data:
   Enter name: McKinley
   Enter age: 22
```



```
Enter another person (y/n)? n

Person:
   Name: Whitney
   Age: 20
Person:
   Name: Rainier
   Age: 21
Person:
   Name: McKinley
   Age: 22
```

```cpp
int main()
 { char ch;
   person pers;       //create person object
   fstream file;      //create input/output file
                      //open for append
   file.open("GROUP.DAT", ios::app | ios::out |
                          ios::in | ios::binary );
   do                 //data from user to file
   {  cout << "\nEnter person's data:";
      pers.getData();  //get one person's data
                       //write to file
      file.write(
reinterpret_cast<char*>(&pers), sizeof(pers) );
      cout << "Enter another person (y/n)? ";
      cin >> ch;
   }while(ch=='y');          //quit on 'n'
   file.seekg(0);     //reset to start of file
                      //read first person
file.read( reinterpret_cast<char*>(&pers),
sizeof(pers) );
   while( !file.eof() )          //quit on EOF
      {
      cout << "\nPerson:";  //display person
      pers.showData();   //read another person
file.read( reinterpret_cast<char*>(&pers),
sizeof(pers) );
      }
   cout << endl;
   return 0;
   }
```

Disk File I/O with Streams (7)

- The Mode Bits

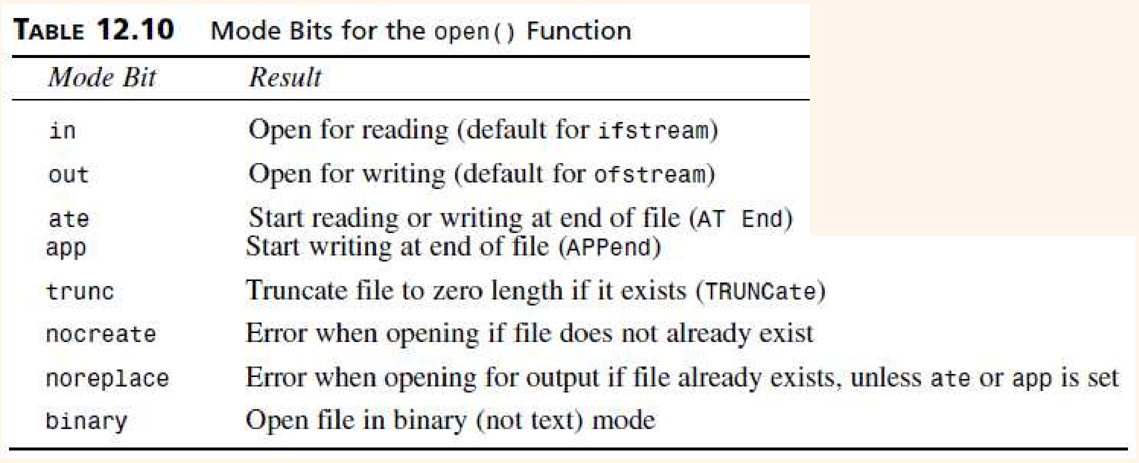
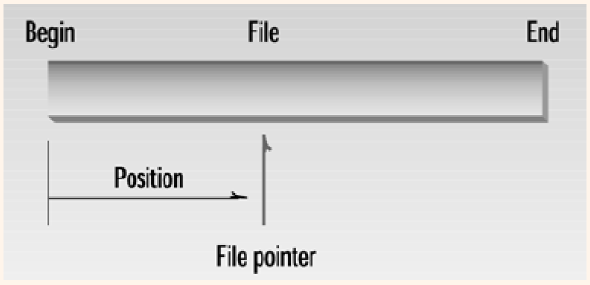


TABLE 12.10 Mode Bits for the open() Function

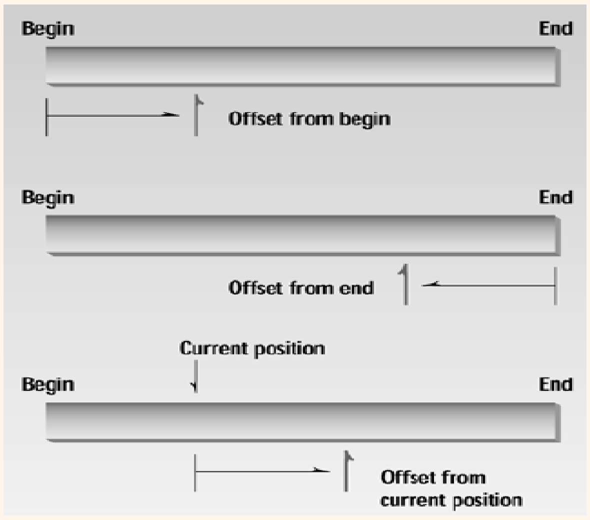
Mode Bit	*Result*
in	Open for reading (default for ifstream)
out	Open for writing (default for ofstream)
ate	Start reading or writing at end of file (AT End)
app	Start writing at end of file (APPend)
trunc	Truncate file to zero length if it exists (TRUNCate)
nocreate	Error when opening if file does not already exist
noreplace	Error when opening for output if file already exists, unless ate or app is set
binary	Open file in binary (not text) mode

File Pointers

- Specifying the Position: seekg()



- Specifying the Offset:



- Get current file pointer position: tellg()

File Pointers (2)

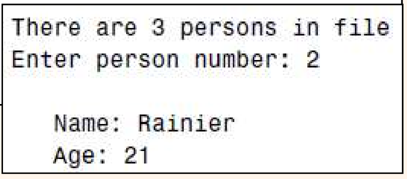
- Specifying the Offset

```
// seekg.cpp
// seeks particular person in file
#include <fstream>          //for file streams
#include <iostream>
using namespace std;
///////////////////////////////////////////////
class person            //class of persons
   {
   protected:
      char name[80];          //person's name
      int age;                //person's age
   public:
      void getData()      //get person's data
         {
         cout << "\n   Enter name: "; cin >>
 name;
         cout << "   Enter age: "; cin >> age;
         }
      void showData(void)//display person's data
         {
         cout << "\n   Name: " << name;
         cout << "\n   Age: " << age;
         }
   };
///////////////////////////////////////////////
```

```
int main()
   {
   person pers;      //create person object
   ifstream infile;  //create input file
   infile.open("GROUP.DAT", ios::in |
ios::binary);  //open file

   infile.seekg(0, ios::end);
          //go to 0 bytes from end
   int endposition = infile.tellg();
          //find where we are
   int n = endposition / sizeof(person);
          //number of persons
cout << "\nThere are " << n << " persons in file";

   cout << "\nEnter person number: ";
   cin >> n;
   int position = (n-1) * sizeof(person);
          //number times size
   infile.seekg(position);//bytes from start
                //read one person
   infile.read( reinterpret_cast<char*>(&pers),
sizeof(pers) );
   pers.showData();    //display the person
   cout << endl;
   return 0;
   }
```



```
There are 3 persons in file
Enter person number: 2

   Name: Rainier
   Age: 21
```