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

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

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`

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`'.

TABLE 12.6 Continued

Function	*Purpose*
`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.

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.

TABLE 12.7 Continued

Function	*Purpose*
`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.

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)`

```
#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
      {
      infile.getline(buffer, MAX);      //read a line of text
      cout << buffer << endl;           //display it
      }
   return 0;
   }
```

The Mode Bits

We've seen the mode bit `ios::binary` before. In the `open()` function we include several new mode bits. The mode bits, defined in `ios`, specify various aspects of how a stream object will be opened. Table 12.10 shows the possibilities.

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`)

TABLE 12.10 Continued

Mode Bit	*Result*
`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

TABLE 12.11 Hardware Device Names

Name	*Device*
`con`	Console (keyboard and screen)
`aux` or `com1`	First serial port
`com2`	Second serial port
`prn` or `lpt1`	First parallel printer

TABLE 12.11 Continued

Name	*Device*
`lpt2`	Second parallel printer
`lpt3`	Third parallel printer
`nul`	Dummy (nonexistent) device