Chapter 19

IOSTREAMS

***Listing 19-1. Wrapping the stdio File Class***

**//: C19:FileClass.h**

**// stdio files wrapped.**

**#ifndef FILECLASS\_H**

**#define FILECLASS\_H**

**#include <cstdio>**

**#include <stdexcept>**

**class FileClass {**

**std::FILE\* f;**

**public:**

**struct FileClassError : std::runtime\_error {**

**FileClassError(const char\* msg)**

**: std::runtime\_error(msg) {}**

**};**

**FileClass(const char\* fname, const char\* mode = "r");**

**~FileClass();**

**std::FILE\* fp();**

**};**

**#endif // FILECLASS\_H ///:~**

***Listing 19-2. Implementing the header file in Listing 19-1***

**//: C19:FileClass.cpp {O}**

**// FileClass Implementation.**

**#include "FileClass.h" // To be INCLUDED from Header FILE above**

**#include <cstdlib>**

**#include <cstdio>**

**using namespace std;**

**FileClass::FileClass(const char\* fname, const char\* mode) {**

**if((f = fopen(fname, mode)) == 0)**

**throw FileClassError("Error opening file");**

**}**

**FileClass::~FileClass() { fclose(f); }**

**FILE\* FileClass::fp() { return f; } ///:~**

***Listing 19-3. Testing Out the Implementation in Listing 19-2***

**//: C19:FileClassTest.cpp**

**//{L} FileClass**

**#include <cstdlib>**

**#include <iostream>**

**#include "FileClass.h"**

**using namespace std;**

**int main() {**

**try {**

**FileClass f("FileClassTest.cpp");**

**const int BSIZE = 100;**

**char buf[BSIZE];**

**while(fgets(buf, BSIZE, f.fp()))**

**fputs(buf, stdout);**

**} catch(FileClass::FileClassError& e) {**

**cout << e.what() << endl;**

**return EXIT\_FAILURE;**

**}**

**return EXIT\_SUCCESS;**

**} // File automatically closed by destructor**

**///:~**

***Listing 19-4. Hiding File I/O in C++***

**//: C19:Fullwrap.h**

**// Completely hidden file I/O.**

**#ifndef FULLWRAP\_H**

**#define FULLWRAP\_H**

**#include <cstddef>**

**#include <cstdio>**

**#undef getc**

**#undef putc**

**#undef ungetc**

**using std::size\_t;**

**using std::fpos\_t;**

**class File {**

**std::FILE\* f;**

**std::FILE\* F(); // Produces checked pointer to f**

**public:**

**File(); // Create object but don't open file**

**File(const char\* path, const char\* mode = "r");**

**~File();**

**int open(const char\* path, const char\* mode = "r");**

**int reopen(const char\* path, const char\* mode);**

**int getc();**

**int ungetc(int c);**

**int putc(int c);**

**int puts(const char\* s);**

**char\* gets(char\* s, int n);**

**int printf(const char\* format, ...);**

**size\_t read(void\* ptr, size\_t size, size\_t n);**

**size\_t write(const void\* ptr, size\_t size, size\_t n);**

**int eof();**

**int close();**

**int flush();**

**int seek(long offset, int whence);**

**int getpos(fpos\_t\* pos);**

**int setpos(const fpos\_t\* pos);**

**long tell();**

**void rewind();**

**void setbuf(char\* buf);**

**int setvbuf(char\* buf, int type, size\_t sz);**

**int error();**

**void clearErr();**

**};**

**#endif // FULLWRAP\_H ///:~**

***Listing 19-5. Date Class Files***

***//:* C19:Date.h**

***#ifndef DATE\_H***

***#define DATE\_H***

***#include <string>***

***#include <stdexcept>***

***#include <iosfwd>***

**class Date {**

**int year, month, day;**

**int compare(const Date&) const;**

**static int daysInPrevMonth(int year, int mon);**

**public:**

***// A class for date calculations***

**struct Duration {**

**int years, months, days;**

**Duration(int y, int m, int d)**

**: years(y), months(m) ,days(d) {}**

**};**

***// An exception class***

**struct DateError : public std::logic\_error {**

**DateError(const std::string& msg = "")**

**: std::logic\_error(msg) {}**

**};**

**Date();**

**Date(int, int, int) throw(DateError);**

**Date(const std::string&) throw(DateError);**

**int getYear() const;**

**int getMonth() const;**

**int getDay() const;**

**std::string toString() const;**

**friend Duration duration(const Date&, const Date&);**

**friend bool operator<(const Date&, const Date&);**

**friend bool operator<=(const Date&, const Date&);**

**friend bool operator>(const Date&, const Date&);**

**friend bool operator>=(const Date&, const Date&);**

**friend bool operator==(const Date&, const Date&);**

**friend bool operator!=(const Date&, const Date&);**

**friend std::ostream& operator<<(std::ostream&,**

**const Date&);**

**friend std::istream& operator>>(std::istream&, Date&);**

**};**

**#endif *// DATE\_H ///:~***

***//:* C19:Date.cpp {O}**

***#include "Date.h" // To be INCLUDED from Header FILE above***

***#include <iostream>***

***#include <sstream>***

***#include <cstdlib>***

***#include <string>***

***#include <algorithm> // For swap()***

***#include <ctime>***

***#include <cassert>***

***#include <iomanip>***

***using namespace std;***

**namespace {**

**const int daysInMonth[][13] = {**

**{ 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 },**

**{ 0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 }**

**};**

**inline bool isleap(int y) {**

**return y%4 == 0 && y%100 != 0 || y%400 == 0;**

**}**

**}**

**Date::Date() {**

***// Get current date***

**time\_t tval = time(0);**

**struct tm \*now = localtime(&tval);**

**year = now->tm\_year + 1900;**

**month = now->tm\_mon + 1;**

**day = now->tm\_mday;**

**}**

**Date::Date(int yr,int mon,int dy) throw(Date::DateError) {**

**if(!(1 <= mon && mon <= 12))**

**throw DateError("Bad month in Date ");**

**if(!(1 <= dy && dy <= daysInMonth[isleap(year)][mon]))**

**throw DateError("Bad day in Date ");**

**year = yr;**

**month = mon;**

**day = dy;**

**}**

**Date::Date(const std::string& s) throw(Date::DateError) {**

***// Assume YYYYMMDD format***

**if(!(s.size() == 8))**

**throw DateError("Bad string in Date ");**

**for(int n = 8; --n >= 0;)**

**if(!isdigit(s[n]))**

**throw DateError("Bad string in Date ");**

**string buf = s.substr(0, 4);**

**year = atoi(buf.c\_str());**

**buf = s.substr(4, 2);**

**month = atoi(buf.c\_str());**

**buf = s.substr(6, 2);**

**day = atoi(buf.c\_str());**

**if(!(1 <= month && month <= 12))**

**throw DateError("Bad month in Date ");**

**if(!(1 <= day && day <=**

**daysInMonth[isleap(year)][month]))**

**throw DateError("Bad day in Date ");**

**}**

**int Date::getYear() const { return year; }**

**int Date::getMonth() const { return month; }**

**int Date::getDay() const { return day; }**

**string Date::toString() const {**

**ostringstream os;**

**os.fill('0');**

**os << setw(4) << year**

**<< setw(2) << month**

**<< setw(2) << day;**

**return os.str();**

**}**

**int Date::compare(const Date& d2) const {**

**int result = year - d2.year;**

**if(result == 0) {**

**result = month - d2.month;**

**if(result == 0)**

**result = day - d2.day;**

**}**

**return result;**

**}**

**int Date::daysInPrevMonth(int year, int month) {**

**if(month == 1) {**

**--year;**

**month = 12;**

**}**

**else**

**--month;**

**return daysInMonth[isleap(year)][month];**

**}**

**bool operator<(const Date& d1, const Date& d2) {**

**return d1.compare(d2) < 0;**

**}**

**bool operator<=(const Date& d1, const Date& d2) {**

**return d1 < d2 || d1 == d2;**

**}**

**bool operator>(const Date& d1, const Date& d2) {**

**return !(d1 < d2) && !(d1 == d2);**

**}**

**bool operator>=(const Date& d1, const Date& d2) {**

**return !(d1 < d2);**

**}**

**bool operator==(const Date& d1, const Date& d2) {**

**return d1.compare(d2) == 0;**

**}**

**bool operator!=(const Date& d1, const Date& d2) {**

**return !(d1 == d2);**

**}**

**Date::Duration**

**duration(const Date& date1, const Date& date2) {**

**int y1 = date1.year;**

**int y2 = date2.year;**

**int m1 = date1.month;**

**int m2 = date2.month;**

**int d1 = date1.day;**

**int d2 = date2.day;**

***// Compute the compare***

**int order = date1.compare(date2);**

**if(order == 0)**

**return Date::Duration(0,0,0);**

**else if(order > 0) {**

***// Make date1 precede date2 locally***

**using std::swap;**

**swap(y1, y2);**

**swap(m1, m2);**

**swap(d1, d2);**

**}**

**int years = y2 - y1;**

**int months = m2 - m1;**

**int days = d2 - d1;**

**assert(years > 0 ||**

**years == 0 && months > 0 ||**

**years == 0 && months == 0 && days > 0);**

***// Do the obvious corrections (must adjust days before months!)***

***// This is a loop in case the previous month is February, and days < -28.***

**int lastMonth = m2;**

**int lastYear = y2;**

**while(days < 0) {**

***// Borrow from month***

**assert(months > 0);**

**days += Date::daysInPrevMonth(**

**lastYear, lastMonth--);**

**--months;**

**}**

**if(months < 0) {**

***// Borrow from year***

**assert(years > 0);**

**months += 12;**

**--years;**

**}**

**return Date::Duration(years, months, days);**

**}**

**ostream& operator<<(ostream& os, const Date& d) {**

**char fillc = os.fill('0');**

**os << setw(2) << d.getMonth() << ‘-‘**

**<< setw(2) << d.getDay() << ‘-‘**

**<< setw(4) << setfill(fillc) << d.getYear();**

**return os;**

**}**

**istream& operator>>(istream& is, Date& d) {**

**is >> d.month;**

**char dash;**

**is >> dash;**

**if(dash != '-')**

**is.setstate(ios::failbit);**

**is >> d.day;**

**is >> dash;**

**if(dash != '-')**

**is.setstate(ios::failbit);**

**is >> d.year;**

**return is;**

**} *///:~***

***Listing 19-6. Illustrating iostream Examples***

**//: C19:Iosexamp.cpp {RunByHand}**

**// Iostream examples.**

**#include <iostream>**

**using namespace std;**

**int main() {**

**int i;**

**cin >> i;**

**float f;**

**cin >> f;**

**char c;**

**cin >> c;**

**char buf[100];**

**cin >> buf;**

**cout << "i = " << i << endl;**

**cout << "f = " << f << endl;**

**cout << "c = " << c << endl;**

**cout << "buf = " << buf << endl;**

**cout << flush;**

**cout << hex << "0x" << i << endl;**

**} ///:~**

***Listing 19-7. Illustrating Stream I/O with Files***

**//: C19:Strfile.cpp**

**// Stream I/O with files;**

**// The difference between get() & getline().**

**#include <fstream>**

**#include <iostream>**

**#include "../require.h"// To be INCLUDED from *Chapter 9***

**using namespace std;**

**int main() {**

**const int SZ = 100; // Buffer size;**

**char buf[SZ];**

**{**

**ifstream in("Strfile.cpp"); // Read**

**assure(in, "Strfile.cpp"); // Verify open**

**ofstream out("Strfile.out"); // Write**

**assure(out, "Strfile.out");**

**int i = 1; // Line counter**

**// A less-convenient approach for line input:**

**while(in.get(buf, SZ)) { // Leaves \n in input**

**in.get(); // Throw away next character (\n)**

**cout << buf << endl; // Must add \n**

**// File output just like standard I/O:**

**out << i++ << ": " << buf << endl;**

**}**

**} // Destructors close in & out**

**ifstream in("Strfile.out");**

**assure(in, "Strfile.out");**

**// More convenient line input:**

**while(in.getline(buf, SZ)) { // Removes \n**

**char\* cp = buf;**

**while(\*cp != ':')**

**++cp;**

**cp += 2; // Past ": "**

**cout << cp << endl;**

**// Must still add \n**

**}**

**} ///:~**

***Listing 19-8. Typing a File to Standard Output***

**//: C19:Stype.cpp**

**// Type a file to standard output.**

**#include <fstream>**

**#include <iostream>**

**#include "../require.h"**

**using namespace std;**

**int main() {**

**ifstream in("Stype.cpp");**

**assure(in, "Stype.cpp");**

**cout << in.rdbuf(); // Outputs entire file**

**} ///:~**

***Listing 19-9. Copying a File to Standard Output***

**//: C19:Sbufget.cpp**

**// Copies a file to standard output.**

**#include <fstream>**

**#include <iostream>**

**#include "../require.h"**

**using namespace std;**

**int main() {**

**ifstream in("Sbufget.cpp");**

**assure(in);**

**streambuf& sb = \*cout.rdbuf();**

**while(!in.get(sb).eof()) {**

**if(in.fail())          // Found blank line**

**in.clear();**

**cout << char(in.get()); // Process '\n'**

**}**

**} ///:~**

***Listing 19-10. Demonstration of Seeking in iostreams***

**//: C19:Seeking.cpp**

**// Seeking in iostreams.**

**#include <cassert>**

**#include <cstddef>**

**#include <cstring>**

**#include <fstream>**

**#include "../require.h"**

**using namespace std;**

**int main() {**

**const int STR\_NUM = 5, STR\_LEN = 30;**

**char origData[STR\_NUM][STR\_LEN] = {**

**"Hickory dickory dus. . .",**

**"Are you tired of C++?",**

**"Well, if you have,",**

**"That's just too bad,",**

**"There's plenty more for us!"**

**};**

**char readData[STR\_NUM][STR\_LEN] = {{ 0 }};**

**ofstream out("Poem.bin", ios::out | ios::binary);**

**assure(out, "Poem.bin");**

**for(int i = 0; i < STR\_NUM; i++)**

**out.write(origData[i], STR\_LEN);**

**out.close();**

**ifstream in("Poem.bin", ios::in | ios::binary);**

**assure(in, "Poem.bin");**

**in.read(readData[0], STR\_LEN);**

**assert(strcmp(readData[0], "Hickory dickory dus. . .")**

**== 0);**

**// Seek -STR\_LEN bytes from the end of file**

**in.seekg(-STR\_LEN, ios::end);**

**in.read(readData[1], STR\_LEN);**

**assert(strcmp(readData[1], "There's plenty more for us!")**

**== 0);**

**// Absolute seek (like using operator[] with a file)**

**in.seekg(3 \* STR\_LEN);**

**in.read(readData[2], STR\_LEN);**

**assert(strcmp(readData[2], "That's just too bad,") == 0);**

**// Seek backwards from current position**

**in.seekg(-STR\_LEN \* 2, ios::cur);**

**in.read(readData[3], STR\_LEN);**

**assert(strcmp(readData[3], "Well, if you have,") == 0);**

**// Seek from the beginning of the file**

**in.seekg(1 \* STR\_LEN, ios::beg);**

**in.read(readData[4], STR\_LEN);**

**assert(strcmp(readData[4], "Are you tired of C++?")**

**== 0);**

**} ///:~**

***Listing 19-11. Demonstration of Reading and Writing One File***

**//: C19:Iofile.cpp**

**// Reading & writing one file.**

**#include <fstream>**

**#include <iostream>**

**#include "../require.h"**

**using namespace std;**

**int main() {**

**ifstream in("Iofile.cpp");**

**assure(in, "Iofile.cpp");**

**ofstream out("Iofile.out");**

**assure(out, "Iofile.out");**

**out << in.rdbuf();**

**// Copy file**

**in.close();**

**out.close();**

**// Open for reading and writing:**

**ifstream in2("Iofile.out", ios::in | ios::out);**

**assure(in2, "Iofile.out");**

**ostream out2(in2.rdbuf());**

**cout << in2.rdbuf();  // Print whole file**

**out2 << "Where does this end up?";**

**out2.seekp(0, ios::beg);**

**out2 << "And what about this?";**

**in2.seekg(0, ios::beg);**

**cout << in2.rdbuf();**

**} ///:~**

***Listing 19-12. Demonstrating Input String Streams***

**//: C19:Istring.cpp**

**// Input string streams.**

**#include <cassert>**

**#include <cmath>  // For fabs()**

**#include <iostream>**

**#include <limits> // For epsilon()**

**#include <sstream>**

**#include <string>**

**using namespace std;**

**int main() {**

**istringstream s("47 1.414 This is a test");**

**int i;**

**double f;**

**s >> i >> f; // Whitespace-delimited input**

**assert(i == 47);**

**double relerr = (fabs(f) - 1.414) / 1.414;**

**assert(relerr <= numeric\_limits<double>::epsilon());**

**string buf2;**

**s >> buf2;**

**assert(buf2 == "This");**

**cout << s.rdbuf(); // " is a test"**

**} ///:~**

***Listing 19-13. Testing the Date Extractor***

**//: C19:DateIOTest.cpp**

**//{L} ../C19/Date**

**#include <iostream>**

**#include <sstream>**

**#include "../Date.h"**

**using namespace std;**

**void testDate(const string& s) {**

**istringstream os(s);**

**Date d;**

**os >> d;**

**if(os)**

**cout << d << endl;**

**else**

**cout << "input error with \"" << s << "\"" << endl;**

**}**

**int main() {**

**testDate("08-10-2003");**

**testDate("8-10-2003");**

**testDate("08 - 10 - 2003");**

**testDate("A-10-2003");**

**testDate("08%10/2003");**

**} ///:~**

***Listing 19-14. Illustrating the Use of ostringstream***

**//: C19:Ostring.cpp {RunByHand}**

**// Illustrates ostringstream.**

**#include <iostream>**

**#include <sstream>**

**#include <string>**

**using namespace std;**

**int main() {**

**cout << "type an int, a float and a string: ";**

**int i;**

**float f;**

**cin >> i >> f;**

**cin >> ws; // Throw away white space**

**string stuff;**

**getline(cin, stuff); // Get rest of the line**

**ostringstream os;**

**os << "integer = " << i << endl;**

**os << "float = " << f << endl;**

**os << "string = " << stuff << endl;**

**string result = os.str();**

**cout << result << endl;**

**} ///:~**

***Listing 19-15. Illustrating A More Elegant HTML Stripper using String Streams***

***(the Previous One, you came across in Listing 18-20: Refer to Chapter 18)***

**//: C19:HTMLStripper2.cpp {RunByHand}**

**//{L} ../C19/ReplaceAll**

**// Filter to remove html tags and markers.**

**#include <cstddef>**

**#include <cstdlib>**

**#include <fstream>**

**#include <iostream>**

**#include <sstream>**

**#include <stdexcept>**

**#include <string>**

**#include "../ReplaceAll.h" // To be INCLUDED from Chapter 18**

**#include "../require.h"**

**using namespace std;**

**string& stripHTMLTags(string& s) throw(runtime\_error) {**

**size\_t leftPos;**

**while((leftPos = s.find('<')) != string::npos) {**

**size\_t rightPos = s.find('>', leftPos+1);**

**if(rightPos == string::npos) {**

**ostringstream msg;**

**msg << "Incomplete HTML tag starting in position "**

**<< leftPos;**

**throw runtime\_error(msg.str());**

**}**

**s.erase(leftPos, rightPos - leftPos + 1);**

**}**

**// Remove all special HTML characters**

**replaceAll(s, "&lt;", "<");**

**replaceAll(s, "&gt;", ">");**

**replaceAll(s, "&amp;", "&");**

**replaceAll(s, "&nbsp;", " ");**

**// Etc...**

**return s;**

**}**

**int main(int argc, char\* argv[]) {**

**requireArgs(argc, 1,**

**"usage: HTMLStripper2 InputFile");**

**ifstream in(argv[1]);**

**assure(in, argv[1]);**

**// Read entire file into string; then strip**

**ostringstream ss;**

**ss << in.rdbuf();**

**try {**

**string s = ss.str();**

**cout << stripHTMLTags(s) << endl;**

**return EXIT\_SUCCESS;**

**} catch(runtime\_error& x) {**

**cout << x.what() << endl;**

**return EXIT\_FAILURE;**

**}**

**} ///:~**

***Listing 19-16. Reading and Writing a String Stream***

**//: C19:StringSeeking.cpp {-bor}{-dmc}**

**// Reads and writes a string stream.**

**#include <cassert>**

**#include <sstream>**

**#include <string>**

**using namespace std;**

**int main() {**

**string text = "We will hook no fish";**

**stringstream ss(text);**

**ss.seekp(0, ios::end);**

**ss << " before its time.";**

**assert(ss.str() ==**

**"We will hook no fish before its time.");**

**// Change "hook" to "ship"**

**ss.seekg(8, ios::beg);**

**string word;**

**ss >> word;**

**assert(word == "hook");**

**ss.seekp(8, ios::beg);**

**ss << "ship";**

**// Change "fish" to "code"**

**ss.seekg(16, ios::beg);**

**ss >> word;**

**assert(word == "fish");**

**ss.seekp(16, ios::beg);**

**ss << "code";**

**assert(ss.str() ==**

**"We will ship no code before its time.");**

**ss.str("A horse of a different color.");**

**assert(ss.str() == "A horse of a different color.");**

**} ///:~**

***Listing 19-17. Illustrating the Use of Unit Buffering***

**//: C19:Unitbuf.cpp {RunByHand}**

**#include <cstdlib>  // For abort()**

**#include <fstream>**

**using namespace std;**

**int main() {**

**ofstream out("log.txt");**

**out.setf(ios::unitbuf);**

**out << "one" << endl;**

**out << "two" << endl;**

**abort();**

**} ///:~**

***Listing 19-18. Illustrating All the Formatting Functions***

**//: C19:Format.cpp**

**// Formatting Functions.**

**#include <fstream>**

**#include <iostream>**

**#include "../require.h"**

**using namespace std;**

**#define D(A) T << #A << endl; A**

**int main() {**

**ofstream T("format.out");**

**assure(T);**

**D(int i = 47;)**

**D(float f = 2300114.414159;)**

**const char\* s = "Is there any more?";**

**D(T.setf(ios::unitbuf);)**

**D(T.setf(ios::showbase);)**

**D(T.setf(ios::uppercase | ios::showpos);)**

**D(T << i << endl;) // Default is dec**

**D(T.setf(ios::hex, ios::basefield);)**

**D(T << i << endl;)**

**D(T.setf(ios::oct, ios::basefield);)**

**D(T << i << endl;)**

**D(T.unsetf(ios::showbase);)**

**D(T.setf(ios::dec, ios::basefield);)**

**D(T.setf(ios::left, ios::adjustfield);)**

**D(T.fill('0');)**

**D(T << "fill char: " << T.fill() << endl;)**

**D(T.width(10);)**

**T << i << endl;**

**D(T.setf(ios::right, ios::adjustfield);)**

**D(T.width(10);)**

**T << i << endl;**

**D(T.setf(ios::internal, ios::adjustfield);)**

**D(T.width(10);)**

**T << i << endl;**

**D(T << i << endl;) // Without width(10)**

**D(T.unsetf(ios::showpos);)**

**D(T.setf(ios::showpoint);)**

**D(T << "prec = " << T.precision() << endl;)**

**D(T.setf(ios::scientific, ios::floatfield);)**

**D(T << endl << f << endl;)**

**D(T.unsetf(ios::uppercase);)**

**D(T << endl << f << endl;)**

**D(T.setf(ios::fixed, ios::floatfield);)**

**D(T << f << endl;)**

**D(T.precision(20);)**

**D(T << "prec = " << T.precision() << endl;)**

**D(T << endl << f << endl;)**

**D(T.setf(ios::scientific, ios::floatfield);)**

**D(T << endl << f << endl;)**

**D(T.setf(ios::fixed, ios::floatfield);)**

**D(T << f << endl;)**

**D(T.width(10);)**

**T << s << endl;**

**D(T.width(40);)**

**T << s << endl;**

**D(T.setf(ios::left, ios::adjustfield);)**

**D(T.width(40);)**

**T << s << endl;**

**} ///:~**

***Listing 19-19. Illustrating the Use of Manipulators***

**//: C19:Manips.cpp**

**// Format.cpp using manipulators.**

**#include <fstream>**

**#include <iomanip>**

**#include <iostream>**

**using namespace std;**

**int main() {**

**ofstream trc("trace.out");**

**int i = 47;**

**float f = 2300114.414159;**

**char\* s = "Is there any more?";**

**trc << setiosflags(ios::unitbuf**

**| ios::showbase | ios::uppercase**

**| ios::showpos);**

**trc << i << endl;**

**trc << hex << i << endl**

**<< oct << i << endl;**

**trc.setf(ios::left, ios::adjustfield);**

**trc << resetiosflags(ios::showbase)**

**<< dec << setfill('0');**

**trc << "fill char: " << trc.fill() << endl;**

**trc << setw(10) << i << endl;**

**trc.setf(ios::right, ios::adjustfield);**

**trc << setw(10) << i << endl;**

**trc.setf(ios::internal, ios::adjustfield);**

**trc << setw(10) << i << endl;**

**trc << i << endl; // Without setw(10)**

**trc << resetiosflags(ios::showpos)**

**<< setiosflags(ios::showpoint)**

**<< "prec = " << trc.precision() << endl;**

**trc.setf(ios::scientific, ios::floatfield);**

**trc << f << resetiosflags(ios::uppercase) << endl;**

**trc.setf(ios::fixed, ios::floatfield);**

**trc << f << endl;**

**trc << f << endl;**

**trc << setprecision(20);**

**trc << "prec = " << trc.precision() << endl;**

**trc << f << endl;**

**trc.setf(ios::scientific, ios::floatfield);**

**trc << f << endl;**

**trc.setf(ios::fixed, ios::floatfield);**

**trc << f << endl;**

**trc << f << endl;**

**trc << setw(10) << s << endl;**

**trc << setw(40) << s << endl;**

**trc.setf(ios::left, ios::adjustfield);**

**trc << setw(40) << s << endl;**

**} ///:~**

***Listing 19-20. Illustrating Limitations of setw with Input***

**//: C19:InputWidth.cpp**

**// Shows limitations of setw with input.**

**#include <cassert>**

**#include <cmath>**

**#include <iomanip>**

**#include <limits>**

**#include <sstream>**

**#include <string>**

**using namespace std;**

**int main() {**

**istringstream is("one 2.34 five");**

**string temp;**

**is >> setw(2) >> temp;**

**assert(temp == "on");**

**is >> setw(2) >> temp;**

**assert(temp == "e");**

**double x;**

**is >> setw(2) >> x;**

**double relerr = fabs(x - 2.34) / x;**

**assert(relerr <= numeric\_limits<double>::epsilon());**

**} ///:~**

***Listing 19-21. Illustrating Creation of a Manipulator***

**//: C19:nl.cpp**

**// Creating a manipulator.**

**#include <iostream>**

**using namespace std;**

**ostream& nl(ostream& os) {**

**return os << '\n';**

**}**

**int main() {**

**cout << "newlines" << nl << "between" << nl**

**<< "each" << nl << "word" << nl;**

**} ///:~*Listing 19-22. Illustrating the Use of Two Effectors***

***(the Former Outputs a Truncated char String while the Latter Prints a Number in Binary Format)***

**//: C19:Effector.cpp**

**##include <cassert>**

**#include <limits> // For max()**

**#include <sstream>**

**#include <string>**

**using namespace std;**

**// Put out a prefix of a string:**

**class Fixw {**

**string str;**

**public:**

**Fixw(const string& s, int width) : str(s, 0, width) {}**

**friend ostream& operator<<(ostream& os, const Fixw& fw) {**

**return os << fw.str;**

**}**

**};**

**// Print a number in binary:**

**typedef unsigned long ulong;**

**class Bin {**

**ulong n;**

**public:**

**Bin(ulong nn) { n = nn; }**

**friend ostream& operator<<(ostream& os, const Bin& b) {**

**const ulong ULMAX = numeric\_limits<ulong>::max();**

**ulong bit = ~(ULMAX >> 1); // Top bit set**

**while(bit) {**

**os << (b.n & bit ? '1' : '0');**

**bit >>= 1;**

**}**

**return os;**

**}**

**};**

**int main() {**

**string words = "Things that make us happy, make us wise";**

**for(int i = words.size(); --i >= 0;) {**

**ostringstream s;**

**s << Fixw(words, i);**

**assert(s.str() == words.substr(0, i));**

**}**

**ostringstream xs, ys;**

**xs << Bin(0xCAFEBABEUL);**

**assert(xs.str() ==**

**"1100""1010""1111""1110""1011""1010""1011""1110");**

**ys << Bin(0x76543210UL);**

**assert(ys.str() ==**

**"0111""0110""0101""0100""0011""0010""0001""0000");**

**} ///:~*Listing 19-23. Testing Files for Conformance***

**//: C19:Cppcheck.cpp**

**// Configures .h & .cpp files to conform to style**

**// standard. Tests existing files for conformance.**

**#include <fstream>**

**#include <sstream>**

**#include <string>**

**#include <cstddef>**

**#include "../require.h"**

**using namespace std;**

**bool startsWith(const string& base, const string& key) {**

**return base.compare(0, key.size(), key) == 0;**

**}**

**void cppCheck(string fileName) {**

**enum bufs { BASE, HEADER, IMPLEMENT, HLINE1, GUARD1,**

**GUARD2, GUARD3, CPPLINE1, INCLUDE, BUFNUM };**

**string part[BUFNUM];**

**part[BASE] = fileName;**

**// Find any '.' in the string:**

**size\_t loc = part[BASE].find('.');**

**if(loc != string::npos)**

**part[BASE].erase(loc); // Strip extension**

**// Force to upper case:**

**for(size\_t i = 0; i < part[BASE].size(); i++)**

**part[BASE][i] = toupper(part[BASE][i]);**

**// Create file names and internal lines:**

**part[HEADER] = part[BASE] + ".h";**

**part[IMPLEMENT] = part[BASE] + ".cpp";**

**part[HLINE1] = "//" ": " + part[HEADER];**

**part[GUARD1] = "#ifndef " + part[BASE] + "\_H";**

**part[GUARD2] = "#define " + part[BASE] + "\_H";**

**part[GUARD3] = "#endif // " + part[BASE] +"\_H";**

**part[CPPLINE1] = string("//") + ": " + part[IMPLEMENT];**

**part[INCLUDE] = "#include \"" + part[HEADER] + "\"";**

**// First, try to open existing files:**

**ifstream existh(part[HEADER].c\_str()),**

**existcpp(part[IMPLEMENT].c\_str());**

**if(!existh) { // Doesn't exist; create it**

**ofstream newheader(part[HEADER].c\_str());**

**assure(newheader, part[HEADER].c\_str());**

**newheader << part[HLINE1] << endl**

**<< part[GUARD1] << endl**

**<< part[GUARD2] << endl << endl**

**<< part[GUARD3] << endl;**

**} else { // Already exists; verify it**

**stringstream hfile; // Write & read**

**ostringstream newheader; // Write**

**hfile << existh.rdbuf();**

**// Check that first three lines conform:**

**bool changed = false;**

**string s;**

**hfile.seekg(0);**

**getline(hfile, s);**

**bool lineUsed = false;**

**// The call to good() is for Microsoft (later too):**

**for(int line = HLINE1; hfile.good() && line <= GUARD2;**

**++line) {**

**if(startsWith(s, part[line])) {**

**newheader << s << endl;**

**lineUsed = true;**

**if(getline(hfile, s))**

**lineUsed = false;**

**} else {**

**newheader << part[line] << endl;**

**changed = true;**

**lineUsed = false;**

**}**

**}**

**// Copy rest of file**

**if(!lineUsed)**

**newheader << s << endl;**

**newheader << hfile.rdbuf();**

**// Check for GUARD3**

**string head = hfile.str();**

**if(head.find(part[GUARD3]) == string::npos) {**

**newheader << part[GUARD3] << endl;**

**changed = true;**

**}**

**// If there were changes, overwrite file:**

**if(changed) {**

**existh.close();**

**ofstream newH(part[HEADER].c\_str());**

**assure(newH, part[HEADER].c\_str());**

**newH << "//@//\n"  // Change marker**

**<< newheader.str();**

**}**

**}**

**if(!existcpp) { // Create cpp file**

**ofstream newcpp(part[IMPLEMENT].c\_str());**

**assure(newcpp, part[IMPLEMENT].c\_str());**

**newcpp << part[CPPLINE1] << endl**

**<< part[INCLUDE] << endl;**

**} else { // Already exists; verify it**

**stringstream cppfile;**

**ostringstream newcpp;**

**cppfile << existcpp.rdbuf();**

**// Check that first two lines conform:**

**bool changed = false;**

**string s;**

**cppfile.seekg(0);**

**getline(cppfile, s);**

**bool lineUsed = false;**

**for(int line = CPPLINE1;**

**cppfile.good() && line <= INCLUDE; ++line) {**

**if(startsWith(s, part[line])) {**

**newcpp << s << endl;**

**lineUsed = true;**

**if(getline(cppfile, s))**

**lineUsed = false;**

**} else {**

**newcpp << part[line] << endl;**

**changed = true;**

**lineUsed = false;**

**}**

**}**

**// Copy rest of file**

**if(!lineUsed)**

**newcpp << s << endl;**

**newcpp << cppfile.rdbuf();**

**// If there were changes, overwrite file:**

**if(changed) {**

**existcpp.close();**

**ofstream newCPP(part[IMPLEMENT].c\_str());**

**assure(newCPP, part[IMPLEMENT].c\_str());**

**newCPP << "//@//\n"  // Change marker**

**<< newcpp.str();**

**}**

**}**

**}**

**int main(int argc, char\* argv[]) {**

**if(argc > 1)**

**cppCheck(argv[1]);**

**else**

**cppCheck("cppCheckTest.h");**

**} ///:~**

***Listing 19-24. Un-commenting Error Generators***

**//: C19:Showerr.cpp {RunByHand}**

**// Uncomment error generators.**

**#include <cstddef>**

**#include <cstdlib>**

**#include <cstdio>**

**#include <fstream>**

**#include <iostream>**

**#include <sstream>**

**#include <string>**

**#include "../require.h"**

**using namespace std;**

**const string USAGE =**

**"usage: showerr filename chapnum\n"**

**"where filename is a C++ source file\n"**

**"and chapnum is the chapter name it's in.\n"**

**"Finds lines commented with //! and removes\n"**

**"the comment, appending //(#) where # is unique\n"**

**"across all files, so you can determine\n"**

**"if your compiler finds the error.\n"**

**"showerr /r\n"**

**"resets the unique counter.";**

**class Showerr {**

**const int CHAP;**

**const string MARKER, FNAME;**

**// File containing error number counter:**

**const string ERRNUM;**

**// File containing error lines:**

**const string ERRFILE;**

**stringstream edited; // Edited file**

**int counter;**

**public:**

**Showerr(const string& f, const string& en,**

**const string& ef, int c)**

**: CHAP(c), MARKER("//!"), FNAME(f), ERRNUM(en),**

**ERRFILE(ef), counter(0) {}**

**void replaceErrors() {**

**ifstream infile(FNAME.c\_str());**

**assure(infile, FNAME.c\_str());**

**ifstream count(ERRNUM.c\_str());**

**if(count) count >> counter;**

**int linecount = 1;**

**string buf;**

**ofstream errlines(ERRFILE.c\_str(), ios::app);**

**assure(errlines, ERRFILE.c\_str());**

**while(getline(infile, buf)) {**

**// Find marker at start of line:**

**size\_t pos = buf.find(MARKER);**

**if(pos != string::npos) {**

**// Erase marker:**

**buf.erase(pos, MARKER.size() + 1);**

**// Append counter & error info:**

**ostringstream out;**

**out << buf << " // (" << ++counter << ") "**

**<< "Chapter " << CHAP**

**<< " File: " << FNAME**

**<< " Line " << linecount << endl;**

**edited << out.str();**

**errlines << out.str(); // Append error file**

**}**

**else**

**edited << buf << "\n"; // Just copy**

**++linecount;**

**}**

**}**

**void saveFiles() {**

**ofstream outfile(FNAME.c\_str()); // Overwrites**

**assure(outfile, FNAME.c\_str());**

**outfile << edited.rdbuf();**

**ofstream count(ERRNUM.c\_str()); // Overwrites**

**assure(count, ERRNUM.c\_str());**

**count << counter; // Save new counter**

**}**

**};**

**int main(int argc, char\* argv[]) {**

**const string ERRCOUNT("../errnum.txt"),**

**ERRFILE("../errlines.txt");**

**requireMinArgs(argc, 1, USAGE.c\_str());**

**if(argv[1][0] == '/' || argv[1][0] == '-') {**

**// Allow for other switches:**

**switch(argv[1][1]) {**

**case 'r': case 'R':**

**cout << "reset counter" << endl;**

**remove(ERRCOUNT.c\_str()); // Delete files**

**remove(ERRFILE.c\_str());**

**return EXIT\_SUCCESS;**

**default:**

**cerr << USAGE << endl;**

**return EXIT\_FAILURE;**

**}**

**}**

**if(argc == 3) {**

**Showerr s(argv[1], ERRCOUNT, ERRFILE, atoi(argv[2]));**

**s.replaceErrors();**

**s.saveFiles();**

**}**

**} ///:~**

***Listing 19-25. Illustrating A Simple Datalogger Record Layout***

**//: C19:DataLogger.h**

**// Datalogger record layout.**

**#ifndef DATALOG\_H**

**#define DATALOG\_H**

**#include <ctime>**

**#include <iosfwd>**

**#include <string>**

**using std::ostream;**

**struct Coord {**

**int deg, min, sec;**

**Coord(int d = 0, int m = 0, int s = 0)**

**: deg(d), min(m), sec(s) {}**

**std::string toString() const;**

**};**

**ostream& operator<<(ostream&, const Coord&);**

**class DataPoint {**

**std::time\_t timestamp; // Time & day**

**Coord latitude, longitude;**

**double depth, temperature;**

**public:**

**DataPoint(std::time\_t ts, const Coord& lat,**

**const Coord& lon, double dep, double temp)**

**: timestamp(ts), latitude(lat), longitude(lon),**

**depth(dep), temperature(temp) {}**

**DataPoint() : timestamp(0), depth(0), temperature(0) {}**

**friend ostream& operator<<(ostream&, const DataPoint&);**

**};**

**#endif // DATALOG\_H ///:~**

***Listing 19-26. Implementing the header file in Listing 19-25 (DataLogger.h)***

**//: C19:DataLogger.cpp {O}**

**// Datapoint implementations.**

**#include "DataLogger.h"// To be INCLUDED from Header FILE above**

**#include <iomanip>**

**#include <iostream>**

**#include <sstream>**

**#include <string>**

**using namespace std;**

**ostream& operator<<(ostream& os, const Coord& c) {**

**return os << c.deg << '\*' << c.min << '\''**

**<< c.sec << '"';**

**}**

**string Coord::toString() const {**

**ostringstream os;**

**os << \*this;**

**return os.str();**

**}**

**ostream& operator<<(ostream& os, const DataPoint& d) {**

**os.setf(ios::fixed, ios::floatfield);**

**char fillc = os.fill('0'); // Pad on left with '0'**

**tm\* tdata = localtime(&d.timestamp);**

**os << setw(2) << tdata->tm\_mon + 1 << '\\'**

**<< setw(2) << tdata->tm\_mday << '\\'**

**<< setw(2) << tdata->tm\_year+1900 << ' '**

**<< setw(2) << tdata->tm\_hour << ':'**

**<< setw(2) << tdata->tm\_min << ':'**

**<< setw(2) << tdata->tm\_sec;**

**os.fill(' '); // Pad on left with ' '**

**streamsize prec = os.precision(4);**

**os << " Lat:"    << setw(9) << d.latitude.toString()**

**<< ", Long:"  << setw(9) << d.longitude.toString()**

**<< ", depth:" << setw(9) << d.depth**

**<< ", temp:"  << setw(9) << d.temperature;**

**os.fill(fillc);**

**os.precision(prec);**

**return os;**

**} ///:~**

***Listing 19-27. Illustrating Generation of Test Data***

***(using write( ) and the DataPoint Inserter)***

**//: C19:Datagen.cpp**

**// Test data generator.**

**//{L} DataLogger**

**#include <cstdlib>**

**#include <ctime>**

**#include <cstring>**

**#include <fstream>**

**#include "DataLogger.h"**

**#include "../require.h"**

**using namespace std;**

**int main() {**

**time\_t timer;**

**srand(time(&timer)); // Seed the random number generator**

**ofstream data("data.txt");**

**assure(data, "data.txt");**

**ofstream bindata("data.bin", ios::binary);**

**assure(bindata, "data.bin");**

**for(int i = 0; i < 100; i++, timer += 55) {**

**// Zero to 199 meters:**

**double newdepth  = rand() % 200;**

**double fraction = rand() % 100 + 1;**

**newdepth += 1.0 / fraction;**

**double newtemp = 150 + rand() % 200; // Kelvin**

**fraction = rand() % 100 + 1;**

**newtemp += 1.0 / fraction;**

**const DataPoint d(timer, Coord(45,20,31),**

**Coord(22,34,18), newdepth,**

**newtemp);**

**data << d << endl;**

**bindata.write(reinterpret\_cast<const char\*>(&d),**

**sizeof(d));**

**}**

**} ///:~**

***Listing 19-28. Scanning and Verifying the Binary Data***

***(Against the Text File Created by Datagen.cpp in Listing 19-27)***

**//: C19:Datascan.cpp**

**//{L} DataLogger**

**#include <fstream>**

**#include <iostream>**

**#include "DataLogger.h"**

**#include "../require.h"**

**using namespace std;**

**int main() {**

**ifstream bindata("data.bin", ios::binary);**

**assure(bindata, "data.bin");**

**DataPoint d;**

**while(bindata.read(reinterpret\_cast<char\*>(&d),**

**sizeof d))**

**cout << d << endl;**

**} ///:~**

***Listing 19-29. Illustrating the Effects of Locales***

**//: C19:Locale.cpp {-g++}{-bor}{-edg} {RunByHand}**

**// Illustrates effects of locales.**

**#include <iostream>**

**#include <locale>**

**using namespace std;**

**int main() {**

**locale def;**

**cout << def.name() << endl;**

**locale current = cout.getloc();**

**cout << current.name() << endl;**

**float val = 1234.56;**

**cout << val << endl;**

**// Change to French/France**

**cout.imbue(locale("french"));**

**current = cout.getloc();**

**cout << current.name() << endl;**

**cout << val << endl;**

**cout << "Enter the literal 7890,12: ";**

**cin.imbue(cout.getloc());**

**cin >> val;**

**cout << val << endl;**

**cout.imbue(def);**

**cout << val << endl;**

**} ///:~**

***Listing 19-30. Illustrating the ‘moneypunct’ Facet***

**//: C19:Facets.cpp {-bor}{-g++}{-mwcc}{-edg}**

**#include <iostream>**

**#include <locale>**

**#include <string>**

**using namespace std;**

**int main() {**

**// Change to French/France**

**locale loc("french");**

**cout.imbue(loc);**

**string currency =**

**use\_facet<moneypunct<char>>(loc).curr\_symbol();**

**char point =**

**use\_facet<moneypunct<char>>(loc).decimal\_point();**

**cout << "I made " << currency << 12.34 << " today!"**

**<< endl;**

**} ///:~**