UEE1303
Objective-Oriented Programming

C++_Lecture 07:

Streams and File Input/Output

C: How to Program 8th ed.

Agenda

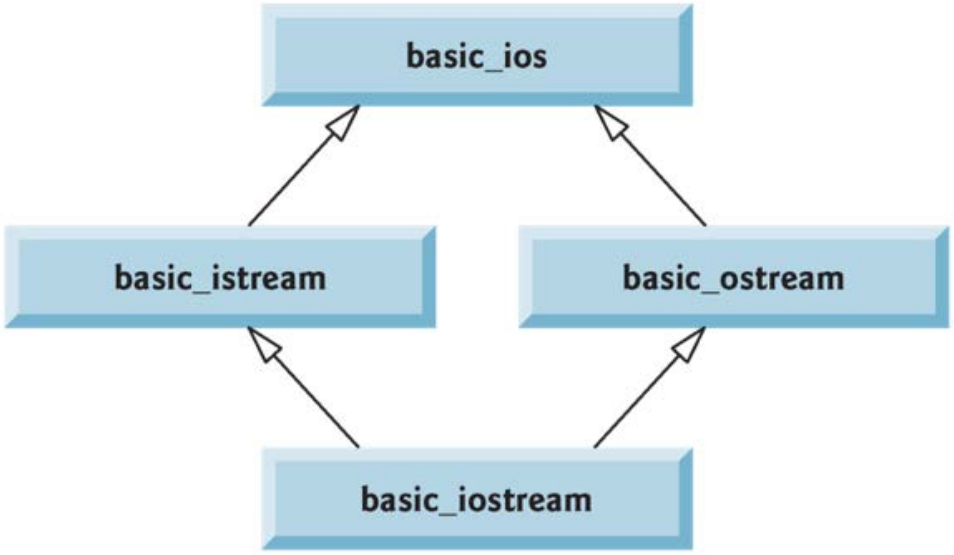
- I/O Stream (Chapter 21.4~21.7)
  - `istream` and `ostream` Member Functions (Chapter 21.3-21.4)
  - Stream Manipulator (Chapter 23.6~23.7) (self-study)
  - Unformatted I/O (Chapter 21.5)
- File Stream
  - Basic Usage
  - Tools: File Names as Input
  - Stream I/O with Files
  - Random Access File

I/O Stream

- A stream is a sequence of bytes used in an input or output operation.
- C++ provides both low-level and high-level input/output (I/O) capabilities
- Low-level I/O is *unformatted*
  - bytes are transferred into and out from memory without regard to the *type of data*
  - for high-volume, high-speed processing
- High-level I/O is formatted
  - bytes are grouped into *meaningful* units such as integers, doubles, and class object types

Stream Class

- The istream class
  - includes a definition of the extraction operator >>
- The ostream class
  - includes a definition of the insertion operator <<
- The iostream is short for input and output stream



Understand `cin` and `cout`

- When you include iostream at the top of your program files, you gain access to these functions `cin` and `cout`
  - `cin` and `cout` are objects and members of the class
  - can use operators such as << and >>
  - polymorphism: can generate different machine instructions when placed with different types of variables

Objects in `<iostream>`

Object	Brief description
`cin`	Object of istream class, connected to the **standard input device**, normally the keyboard.
`cout`	Object of ostream class, connected to **standard output device**, normally the display/screen.
`cerr`	Object of the ostream class connected to **standard error device**. This is unbuffered output, so each insertion to cerr causes its output to appear immediately.
`clog`	Same as cerr but outputs to clog are buffered.

Stream Output

- Formatted and unformatted output capabilities are provided by `ostream`.
- `cout` is a object of the `ostream` class, which supports member functions and overloaded operators
  - Example: the operator<<() function
- Other ostream member functions include `put()`, `flush()`, `eof()`, `bad()`.
- Besides member functions, the `cout` object also has data members, or states.

Ref: http://www.cplusplus.com/reference/ostream/ostream/

put() Function

- Outputs one character at a time
- Examples:

```
cout.put('a');
```

=> outputs letter a to screen

```
cout.put('A').put('\n');
```

=> outputs letter A followed by a newline charter

```
char mystr[10] = "Hello";
cout.put(mystr[1]);
```

=> outputs letter e to screen

Ref: http://www.cplusplus.com/reference/ostream/ostream/put/

Member Functions of `ostream`

`ostream` function	prototype	purpose
`setf()`	`fmtflags setf(fmtflags);`	tasks arguments that set the bits of `cout`
`unsetf()`	`fmtflags unsetf(fmtflags);`	tasks arguments that unset the bits of `cout`
`precision()`	`int precision(int);`	set precision
`width()`	`int width(int);`	set field width

Ref: http://www.cplusplus.com/reference/ostream/ostream/

`setf()` and `unsetf()` Functions

- `setf()` function sets the stream's *format flags* whose bits are set in *fmtfl*, leaving unchanged the rest.

```
fmtflags setf (fmtflags fmtfl);
```

  - The arguments that determine the state of the `cout` object are called format flags or state flags
    - `ios::left` - left-justifies output within the field size, which may be set by the `width()` function
    - `ios::dec` - formats numbers in decimal (base 10)
    - `ios::showpos` - inserts + before positive numbers
    - `ios::showpoint` - displays the decimal point and six significant digits for all floating-point numbers

Ref: http://www.cplusplus.com/reference/ios/ios_base/setf/

setf() and unsetf() Functions (cont.)

```
// modifying flags with setf/unsetf
#include <iostream>
using namespace std;

int main () {
    cout.setf( ios::showpos | ios::dec | ios::showpoint );
    cout << 4.9 << '\n';
    cout.unsetf( ios::showpos );
    cout << 4.9 << '\n';
    cout.unsetf( ios::showpoint );
    cout << 4.9 << '\n';
    return 0;
}
```

```
+4.90000
4.90000
4.9
```

precision() Function

```
// modify precision
#include <iostream>
using namespace std;
int main () {
    double f = 3.14159;
    // floatfield not set
    cout.unsetf ( ios::floatfield );
    cout.precision(5);      cout << f << '\n';
    cout.precision(10);     cout << f << '\n';
    // floatfield set to fixed
    cout.setf( ios::fixed, ios::floatfield );
    cout << f << '\n';
    return 0;
}
```

```
3.1416
3.14159
3.1415900000
```

width() Function

```
// modify precision
#include <iostream>
using namespace std;

int main () {
    cout << 100 << '\n';
    cout.width(10);
    cout << 100 << '\n';
    cout.fill('x');
    cout.width(15);
    cout << std::left << 100 << '\n';
    return 0;
}
```

```
100
       100
100xxxxxxxxxxxx
```

Stream Input

- In C++, the easiest way to read a character is to use `cin` with the extraction operator
  - Extraction operator is an overloaded function named operator>>()

```
cin >>  someValue;
```

- Besides the overloaded extraction operator, including `iostream` provides other input functions
  - most compilers support other istream member functions, such as `eof()`, `bad()`, and `good()`.

Member Functions of `istream`

- As an object, `cin` contains member functions

`istream` function	prototype	purpose

`get()`	`istream& get(char &);`
	`int get();`
	`istream& get(char *str, int len, char c='\n');`
		extract unformatted data from a stream

`getline()`	`istream& getline(char *str, int len, char c = '\n');`	get a line of data from an input stream
`ignore()`	`istream& ignore(int length = 1, char c = '\n');`	extract and discard characters

Ref: http://www.cplusplus.com/reference/istream/istream/

`get()` Function (1)

- The `get()` function takes a character argument and returns a reference to the object that invoked it
  - prototype:

```
istream& get(char &);
```

  - multiple `get()` function calls can be chained

```
char first, middle, last;
cin.get(first);
cin.get(middle).get(last);
```

Ref: http://www.cplusplus.com/reference/istream/istream/get/

`get()` Function (2)

- `cin.get()` can retrieve any character, including letters, numbers, punctuation, and white space such as the character generated by pressing the **Enter** key
- Most compilers overload `get()` so that it can also take no argument
  - Prototype:

```
int get();
```

  - Example:

```
cout << "Press any key to continue";
c = cin.get(); //c: casted to an integer
```

get() Function (2) (cont.)

```
// Fig. 21.4: Fig21_04.cpp
// Using member functions get, put and eof.
#include <iostream>
using namespace std;

int main()
{
    int character;
    // use int, because char cannot represent EOF
    cout << "Before input, cin.eof() is "
         << cin.eof() << endl
         << "Enter a sentence followed by end-of-file:"
         << endl;
```

get() Function (2) (cont.)

```
    // use get to read each character;
    // use put to display it
    while ( ( character = cin.get() ) != EOF )
        cout.put( character );

    // display end-of-file character
    cout << "\nEOF in this system is: " << character
         << endl;
    cout << "After input of EOF, cin.eof() is "
         << cin.eof() << endl;
} // end main
```

```
Before input, cin.eof() is 0
Enter a sentence followed by end-of-file:
Testing the get and put member functions
Testing the get and put member functions
^Z

EOF in this system is: -1
After input of EOF, cin.eof() is 1
```

`get()` Function (3)

- The `istream` class `get()` function is overloaded so that it can take two or three arguments to allow you to input a string of characters.

```
istream& get(char *str, int len,
             char c = '\n');
```

  - *1st* argument is a pointer that holds the address of the string (necessary)
  - *2nd* argument is the number of characters that will be stored (necessary)
  - *3rd* argument is the character that terminates the entry, often called the *delimiter* character

get() Function (3) (cont.)

```
// Fig. 21.5: Fig21_05.cpp
// Contrasting input of a string via cin and cin.get.
#include <iostream>
using namespace std;

int main()
{
    // create two char arrays, each with 80 elements
    const int SIZE = 80;
    char buffer1[ SIZE ];
    char buffer2[ SIZE ];

    // use cin to input characters into buffer1
    cout << "Enter a sentence:" << endl;
    cin >> buffer1;
    // display buffer1 contents
    cout << "\nThe string read with cin was:" << endl
         << buffer1 << endl << endl;
```

get() Function (3) (cont.)

```
     // use cin.get to input characters into buffer2
    cin.get( buffer2, SIZE );
    // display buffer2 contents
    cout << "The string read with cin.get was:" << endl
         << buffer2 << endl;
} // end main
```

```
Enter a sentence:
Contrasting string input with cin and cin.get

The string read with cin was:
Contrasting

The string read with cin.get was:
 string input with cin and cin.get
```

getline() Function

- The `getline()` function reads a line of text at the address represented by `str`.

```
istream& getline(char *str, int len,
                 char c = '\n');
```

  - It reads until it reaches either the length or the character used as the third argument.

Ref: http://www.cplusplus.com/reference/istream/istream/getline/

getline() Function (cont.)

```
// Fig. 21.6: Fig21_06.cpp
// Inputting characters using cin member function getline.
#include <iostream>
using namespace std;

int main()
{
    const int SIZE = 80;
    char buffer[ SIZE ]; // create array of 80 characters

    // input characters in buffer via cin function getline
    cout << "Enter a sentence:" << endl;
    cin.getline( buffer, SIZE );

    // display buffer contents
    cout << "\nThe sentence entered is:" << endl
         << buffer << endl;
} // end main
```

More Member Functions

- `putback()`
  - once read, might need to put back
  - Ex: `cin.putback(lastChar);`
- `peek()`
  - return next char, but leaves it there
  - Ex: `peekChar = cin.peek();`
- `ignore()`
  - skip input, up to designated character
  - Ex: `cin.ignore(1000,'\n');`
    - skip at most 1000 characters until '\n'

putback() Function

```
#include <iostream>
#include <string>
using namespace std;
int main () {
    cout << "Please, enter a number or a word: ";
    char c = cin.get();
    if ( (c >= '0') && (c <= '9') ) {
        int n;
        cin.putback (c);  cin >> n;
        cout << "You entered a number: " << n << '\n';
    }
    else  {
        string str;
        cin.putback (c);  getline (cin, str);
        cout << "You entered a word: " << str << '\n';
    }
    return 0;
}
```

Unformatted I/O

- Unformatted I/O is performed using the read and write member functions of `istream` and `ostream`, respectively.
  - Member function `read` inputs bytes to a character array in member.
  - Member function `write` outputs bytes from a character array.

```
char buffer[] = "Happy Birthday";
cout.write( buffer, 10 );
cout.write( "ABCDEFGHIJKLMNOPQRST", 10);
```

read() and write() Function

```
/ Fig. 21.7: Fig21_07.cpp
// Unformatted I/O using read, gcount and write.
#include <iostream>
using namespace std;
int main()
{
    const int SIZE = 80;
    char buffer[ SIZE ]; // create array of 80 characters
    // use function read to input characters into buffer
    cout << "Enter a sentence:" << endl;
    cin.read( buffer, 20 );

    // use functions write and gcount to display buffer
characters
    cout << endl << "The sentence entered was:" << endl;
    cout.write( buffer, cin.gcount() );
    cout << endl;
} // end main
```

Files

- File: collection of data that is stored together under common name on storage media
  - C++ sources as text files on hard disks
- Reading from file
  - When program takes input
- Writing to file
  - When program sends output
- Start at beginning of file to end
  - Other methods available

File Names

- Files have two names to our programs
- External filename
  - Also called *physical filename*
  - Example: `"input_file.txt"`
  - Sometimes considered *real filename*
  - Used only once in program (to open)
- Stream name
  - Also called *logical filename*
  - Example: `inFile` in `"ifstream inFile;"`
  - C++ program uses this name for all file activity

File Connection & File I/O

- Must first connect file to stream object
  - For input: file => `ifstream` object
  - For output: file => `ofstream` object
- Use `ifstream` and `ofstream` classes

```
#include <fstream>
using namespace std;
```

  - Defined in library `<fstream>`
  - Named in `std` namespace
- Alternative form

```
#include <fstream>
using std::ifstream;
using std::ofstream;
```

Declaring Streams

- Stream must be declared like any other class variable:

```
ifstream inFile;
ofstream outFile;
```

- Must then connect to file:

```
inFile.open("input_file.txt");
```

  - opening the file by member function `open()`
  - can specify complete pathname
  - filename must be a C-String
- Can specify filename at declaration

```
ifstream inFile("input_file.txt");
ofstream outFile("output_file.txt");
```

File Streams Usage

- Once declared => use normally!

```
int oneNumber, anotherNumber;
inFile >> oneNumber >> anotherNumber;
```

- Output stream similar:

```
ofstream outFile;
outFile.open("output_file.txt");
outFile << "oneNumber = "
        << oneNumber
        << "anotherNumber = "
        << anotherNumber;
```

- Send items to output file

Open File Stream w/ Flags

Mode	Description
`ios::app`	*Append* all output to the end of the file.
`ios::ate`	Open a file for output and move to the end of the file (normally used to append data to a file). Data can be written *anywhere* in the file.
`ios::in`	Open a file for *input*.
`ios::out`	Open a file for *output*.
`ios::trunc`	*Discard* the file's contents (this also is the default action for `ios::out`).
`ios::binary`	Open a file for binary, i.e., *nontext*, input or output.

Closing Files

- Files should be closed
  - When program completed getting input or sending output
  - Disconnects stream from file
- Example:

```
inFile.close();
outFile.close();
```

  - no argument
- Files automatically close when program ends
  - Good to close opened files explicitly

Appending to a File

- Standard open operation begins with empty file
  - Even if file exists => contents *lost*
- Open for append:

```
ofstream outFile;
outFile.open("output.txt",ios::app);
```

  - If file doesn't exist =>creates it
  - If file exists => appends to end
  - 2nd argument is class `ios` defined constant

  => in `<iostream>` library, `std` namespace

Checking File Open Success

- File opens could fail
  - If input file doesn't exist
  - No write permissions to output file
- Place call to `.fail()` or `.is_open()` to check stream operation success

```
inFile.open("stuff.txt");
if (inFile.fail())
{
    cout << "File open failed.\n";
    exit(1);
}
```

  - `.is_open()` returns the opposite `.fail()`

Checking End-of-File w/ `eof()`

- Use loop to process file until end
  - two ways to test for the end of file
- Use member function `eof()`

```
inFile.get(next);
while (! inFile.eof())
{
    cout << next;
    inFile.get(next);
}
```

  - Reads each character until file ends
  - `eof()` member function returns `bool`

Checking End-of-File w/ Read

- Second method: read operation returns `bool` value!
  => a good way to read file

  `(inFile >> next)`
  - expression returns `true` if read successful
  - return `false` if attempt to read beyond end of file
- In action:

```
double next, sum = 0;
while (inFile >> next)
    sum = sum + next;
cout << "the sum is " << sum << endl;
```

An Example for File Input

```
// example of file input
#include <fstream>
#include <iostream>
using namespace std;

int main() {
    double sum = 0, t; int count=0;
    ifstream inFile("data.txt", ios::in);
    if (!inFile) { // if (inFile.fail())
        cout << "Cannot open file!" << endl;
        exit (1);
    }
    while (inFile >> t) {
        sum += t; count++;
    }
    cout << "avg=" << sum/count << endl;
    return 0;
}
```

Checking File I/O Status

Prototype	Brief description
`fail()`	Return true if the file has not been opened successfully; otherwise, return false
`eof()`	return true if a read has been attempted past the end-of-file; otherwise, return false
`good()`	return true if the file is available for program use; otherwise, return false
`bad()`	return true if a fatal error with the current stream has occurred; otherwise, false. not normally occur.

Tools: File Names as Input

- Stream open operation
  - Argument to `open()` is of C-String type
  - Can be literal (used so far) or variable

```
string fileName;
ifstream inFile;
cout << "Enter file name: ";
cin >> fileName;
inFile.open(fileName.c_str());
```

  - Provides more flexibility
- Open file from command line argument
  - Ex: `inFile.open(argv[2]);`

Stream I/O with Files

- All `cin` and `cout` character I/O same for files!
- Common character I/O functions
  - `get()`, `getline()`: obtain characters from input file
  - `put()`: put one character to output streams
  - `putback()`: put back the character just read to input streams
  - `peek()`: return the next character from the stream without removing it
  - `ignore()`: skips

get(): Read Characters from File

- get(): obtain characters from file and save it to the input stream. 3 forms:

```
istream& get(char& ch); // most suggested
istream& get(char* buffer, streamsize num);
istream& get(char* buffer, streamsize num,
char delim);
```

- Example for Form 1:

```
ifstream inFile("input_file.dat", ios::in);
char ich;
while (inFile.get(ich)) {
    cout << ich;
}
```

`getline()`: Read a Line from File

- `getline()`: read characters into input stream buffer until either:
  - (num - 1) characters have been read,
  - an EOF is encountered,
  - or, until the character delim (normally, newline, '\n') is read. The delim character is not put into buffer.
- Two forms:

```
istream& getline(char* buffer, streamsize num);
istream& getline(char* buffer, streamsize num, char delim);
```

Examples of `getline()`

- Example 1 (for C-String variables):

```
int MAX_LENGTH = 100;
char line[MAX_LENGTH];
while (inFile.getline(line,MAX_LENGTH)) {
    cout << "read line: " << line << endl;
}
```

- Example 2 (for string variables):

```
string line;
while (getline(inFile, line)) {
    cout << "read line: " << line << endl;
}
```

put( ): Put One Character to File

- `put( )`: put one character to the output stream and save it to the file
- Syntax: `ostream& put(char ch);`

- Example:

```
ofstream outFile("output.txt");
string article = "Today is June-3-2014\n";
for (int i = 0; i < article.size(); i++)
    outFile.put(article[i]);
outFile.close();
```

putback( ): Put Back One Character

- `putback()`: return the previously-read character `ch` to the input stream
- Syntax: `istream& putback(char ch);`
- Example:

```
ifstream inFile("sample.txt");
int n = 0; char str[256];
char c = inFile.get();
if ((c >= '0')&&(c <= '9')) {
    inFile.putback(c);
    inFile >> n;
} else {
    inFile.putback(c);
    inFile >> str;
}
```

peek( ): Read/Return Next Character

- peek( ): return the *next* character in the stream or EOF if the end of file is read
  - not remove the character from the stream
- Syntax: `int peek();`
- Example:

```
ifstream inFile("sample.txt");
int n = 0; char str[256];
char c = inFile.peek();
if ((c >= '0')&&(c <= '9')) {
    inFile >> n;
} else {
    inFile >> str;
}
```

ignore(): Skip Characters

- `ignore()`: read and throw away characters until *num* characters have been read or until the character *delim* is read
- Syntax: `istream& ignore(streamsize num=1, int delim=EOF);`
- Example:

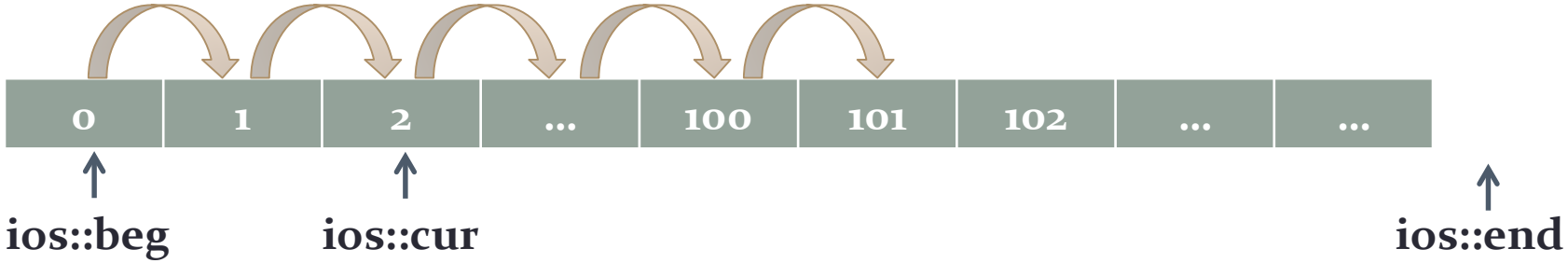
```
char first, last;
cout << "Enter your first and last names:";
first = inFile.get();
inFile.ignore(256,' ');
last = inFile.get();
cout <<"Your initials are " << first << last;
```

Random File Access

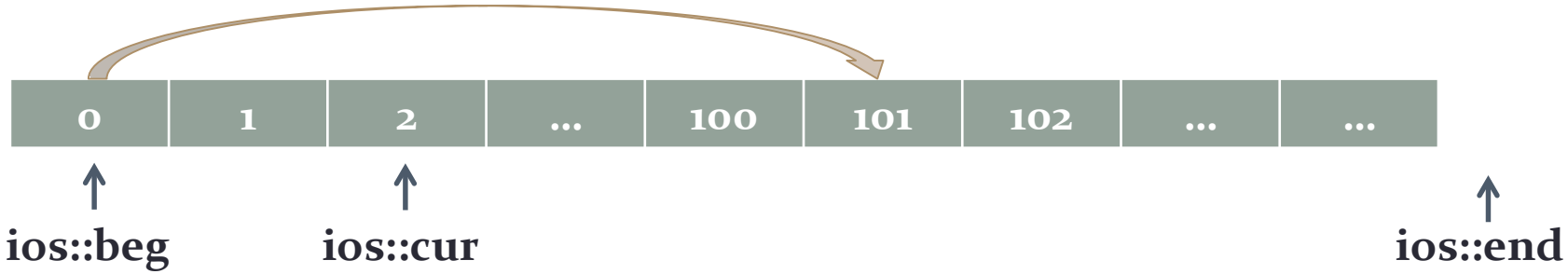
- Sequential file organization: characters in file are stored in sequential manner
- Random Access: any character in an opened file can be read directly without having to read characters ahead of it
- File position marker: long integer that represents an offset from the beginning of each file
  - Keep track of where next character is to be read from or written to
  - Allow for random access of any individual character

Random File Access (cont.)

- Finding record 101 using sequential access:
  - One file = a sequential stream of n characters



- Finding record 101 using random access:



Random File Access (cont.)

Name	Brief description
`seekg(offset, mode)`	For input files, move to the offset position as indicated by the mode
`seekp(offset, mode)`	For output files, move to the offset position as indicated by the mode
`tellg(void)`	For input files, return the current value of the file position marker
`tellp(void)`	For output files, return the current value of the file position marker

- Type of modes:
  - `ios::beg`: the beginning of the file
  - `ios::cur`: current position of the file
  - `ios::end`: the end of the file

Random Access Tools

- Opens same as `istream` or `ostream`

```
fstream rwStream;
rwStream.open("sample.dat",ios::in|ios::out);
```

- Move about in file
  - Positions put-pointer at 1000th byte

    `rwStream.seekp(1000);`
  - Positions get-pointer at 1000th byte

    `rwStream.seekg(1000);`
  - Position put-pointer at 100th record of objects

    `rwStream.seekp(100*sizeof(myStruct)-1);`

Random Access Tools (cont.)

- `seekg()` and `seekp()` can be used with three modes: `ios::beg`, `ios::cur` and `ios::end`
- (EX1) `infile.seekg(10, ios::beg)`
  - File position maker moves to the 10th character from the begging of the file
- (EX2) `infile.seekg(-6, ios::cur)`
  - File position maker moves back 6 character from the current position
- (EX3) `outfile.seekp(0, ios::end)`
  - File position marker moves to the last characters at the end of the file

Summary

- `cout` and `cin` are members of a class
  - >> is overloaded to input all built-in types
- The `ostream` class provides useful output functions: `setf()`, `unsetf()`, `width()`, and `precision()`.
- To perform file output, you must instantiate your own member of the `fstream` or `ofstream` class.

Summary (cont.)

- All file streams must be declared as objects of either the `ifstream` or `ofstream` classes
- Data files can be accessed randomly using the `seekg()`, `seekp()`, `tellg()`,and `tellp()` methods
  - `g` versions of these functions are used to alter and query file position marker for input file streams
  - `p` versions do the same for output file streams

References

- Paul Deitel and Harvey Deitel, “C How to Program” Sixth Edition
  - Chapter 21
- Paul Deitel and Harvey Deitel, “C++ How to Program (late objects version)” Seventh Edition
  - Chapter 8, 15, 17
- W. Savitch, “Absolute C++,” Fourth Edition
  - Chapter 12