More File Operations

Lecture 33
COP 3014 Spring 2017

April 10, 2017

eof() member function

- A useful member function of the input stream classes is eof()
  - Stands for end of file
  - Returns a bool value, answering the question "Are we at the end of the file?" (or is the "end-of-file" character the next one on the stream?)
  - Can be used to indicate whether the end of an input file has been reached, when reading sequentially
- Very useful when reading files where the size of the file or the amount of data to be read is not known in advance

```
while (!in1.eof()) // while not at end of file
{
        // read and process input from the file
}
```

- Can also be used with cin, where the user types a key combination representing the "end-of-file" character
  - On Unix and Mac systems, type ctrl-d to enter the end-of-file character
  - On Windows, type ctrl-z to enter the end-of-file character

Character I/O - Output

- We've already used the insertion operator to print characters:
```
char letter = 'A';
cout << letter;
```
- There is also a member function (of output stream classes) called put(), which can be used to print a character. It's prototype is:
```
ostream& put(char c);
```
- Sample calls:
```
char ch1 = 'A', ch2 = 'B', ch3 = 'C';
cout.put(ch1); // equivalent to:  cout << ch1;
cout.put(ch2); // equivalent to:  cout << ch2;
```
- It can be cascaded, like the insertion operator:
```
cout.put(ch1).put(ch2).put(ch3);
```
- The put() function doesn't really do anything more special than the insertion operator does. It's just listed here for completeness

Character I/O- Input

- There are many versions of the extraction operator $>>$, for reading data from an input stream. This includes a version that reads characters:

```
char letter;
cin >> letter;
```

- However, if we, for example, tried to copy a file into another by reading one character at a time, the output file wouldn't have any whitespace.
- All built-in versions of the extraction operator for input streams will ignore leading white space by default

Character I/O- Input

Here are some other useful member functions (of input stream classes) for working with the input of characters:

- **peek()** – this function returns the ascii value of the next character on the input stream, but does not extract it
- **get()** – the two get functions both extract the next single character on the input stream, and they do not skip any white space.
  - The version with no parameters returns the ascii value of the extracted character
  - The version with the single parameter stores the character in the parameter, passed by reference. Returns a reference to the stream object (or 0, for end-of-file)
- **ignore()** member function - skips either a designated number of characters, or skips up to a specified delimiter.
- **putback()** member function - puts a character back into the input stream

Examples

```
char ch1, ch2, ch3;
cin >> ch1 >> ch2 >> ch3; // reads three characters,
skipping white space

//get():  no parameters, no white space skipped
ch1 = cin.get();
ch2 = cin.get();
ch3 = cin.get();
//get():  one parameter, can be cascaded
cin.get(ch1).get(ch2).get(ch3);

//peek():  trying to read a digit, as a char
char temp = cin.peek(); // look at next character
if (temp < '0' || temp > '9')
     cout << "Not a digit";
else ch1 = cin.get(); // read the digit
```

Passing Stream Objects into Functions

- In a function prototype, any type can be used as a formal parameter type or as a return type.
  - This includes classes, which are programmer-defined types
- Streams can be passed into functions as parameters (and/or returned).
  - Because of how the stream classes were set up, they can only be passed by reference, however
- So, for instance, the following can be return types or parameter types in a function:

  `ostream &`

  `istream &`

  `ofstream &`

  `ifstream &`
- Why? – functions that do output can be written that are more versatile, by allowing the output to go to a variety of places

Passing Stream Objects into Functions

- Example of a more limited function:

```
void Show()
{
      cout << "Hello, World\n";
}
```

- A call to this function always prints to standard output (cout).
- Same function, more versatile:

```
void Show(ostream& output)
{
      output << "Hello, World\n";
}
```

- We can do the printing to different output destinations now:

```
Show(cout); // prints to standard output stream
Show(cerr); // prints to standard error stream
```

Passing Stream Objects into Functions

- This works with file stream types, too:

```
void PrintRecord(ofstream& fout, int acctID,
double balance)
{
     fout << acctID << balance << '\n';
}
```

- Now, we can call this function to print the same data format to different files:

```
ofstream out1, out2;
out1.open("file1.txt");
out2.open("file2.txt");
PrintRecord(out1, 12, 45.6); //print to file1
PrintRecord(out1, 124, 67.89); // print to file1
PrintRecord(out2, 100, 123.09); // print to file2
PrintRecord(out2, 11, 287.64); // print to file2
```