End-of-file Tests and Looping Summary

Larry Caretto
Computer Science 106

Computing in Engineering and Science

March 16, 2006

California State University Northridge

Outline

- Review looping code
- End-of-file tests
  - Files as objects
  - Functions associated with file names
    - .good(), .fail() and .eof()
  - Sentinels versus end-of-file functions
- Practice writing loops

Review Previous Material on Files

- File input/output
  - Have to use #include <fstream> library
  - Associate a program file name with an operating system file name
    - `ifstream inFile( "input.dat" );`
  - Use program file name in place of cin
    - `inFile >> x >> y;`
  - Want to be able to have computer find end of large data file
    - Can use "sentinel" (data item that is not data) to mark end of data stream
    - Alternative is end-of-file test

File Names and Properties

- Program file names are objects
  - Object-oriented programming
  - Objects have functions that operate on specific objects (e.g. ifstream inFile)
  - Usual format is <*object name*>.<*function*>
  - For the declaration ifstream inFile the following functions are applicable
    - inFile.good() is true if previous reads have not found an error or an end of file
    - inFile.fail() is true if a read attempt failed
    - inFile.eof() is true if an end of file is found

Testing for End of File <EOF>

- Consider effects of three separate functions: .good(), .fail() and .eof()
  - .good() is true if a future read statement may be possible (no error or end of file found yet)
  - .fail() is true if a read statement could not be completed (some variables not read)
  - .eof() is true if an end of file is found
- Where is the eof located?
  - Important to understand EOF test

Possible <EOF> Locations

Example 1: 12 14 -23.2<EOF>
Example 2: 12 14 -23.2 <EOF>
Example 3: 12 14 -23.2<newline>
<EOF>

- Example 1 file is saved immediately after the last digit is entered
- Example 2 file has spaces (but no new-line after the last digit
- Example 3 file has <newline> (and possible spaces) after last digit

Result of in >> x >> y >> z;

Example 1: 12 14 -23.2<EOF>
Example 2: 12 14 -23.2 <EOF>
Example 3: 12 14 -23.2<newline>
<EOF>

Function results for each sample file

Function:	in.good()	in.fail()	in.eof()
Example 1:	false	false	true
Example 2:	true	false	false
Example 3:	true	false	false

End of File Test for Sum

```
int x, sum = 0, n = 0;
inFile >> x
while ( !infile.fail() )
{
    sum += x; // sum = sum + x
    n++;
    inFile >> x;
}
if (n != 0) double average =
            double( sum ) / n;
```


End-of-file Exercise

- Read all the data from a file and determine the maximum, minimum, and number of data items on the file
- Hints
  - Use code similar to that on the last chart
  - Read the initial value and set the current minimum and maximum to that value
  - In the loop check each data item against the current minimum and maximum

End of File Exercise Solution

```
double x, xMin, xMax;
int count = 0;
ifstream inFile ( “input.dat” );
inFile >> x;
xMin = x;
xMax = x;
while ( !infile.fail() )
{
    count++;
    if ( x > xMax )
        xMax = x;
    else if ( x < xMin )
        xMin = x;
    inFile >> x;
}
```


File Buffering

- Input and output information is placed in a buffer and transferred from input to code or code to output later
- Input transfer occurs when user presses the enter key
- If not all characters are read, the remaining characters are kept on the input buffer
  - Source of funny input results we saw in exercise two

Testing File Status

- The result of an input operation, say cin >> x >> y, can be tested
  - It is true if there were no errors and no characters left in the input buffer
  - Sample code: `if ( !( cin >> x ) )`
  - Sample code is true if there is an error condition
  - We can use this test to correct any possible errors, including clearing the input buffer

Keyboard Error Test Loop

```
do
{   cout << "Enter x: ";
    bool goodInput = ( cin >> x );
    if ( !goodInput )
    {
       cout << "\nInput error\n";
       cin.clear(); // reset error
       cin.ignore(80,'\n'); //remove
      // bad characters from buffer
    }
} while( !goodInput );
```


Looping Summary

- Structures to repeat code statements
- Use condition in while or do while
- Use for loop for count controlled loop
  - Can actually have complex conditions in for loop for C++ (see exercise six)
- Can use combination operators such as x += 3 and count-- in for loops
- Beware of off-by-one errors in limits use use of < *versus* <= (or > *versus* >=)

Trajectory Exercise

- In the first quiz we saw the formulas for a projectile shot from the ground with a velocity v_0
  - Maximum height, $h_{max} = v_0^2/g$
  - Time to maximum height, $t_{max} = v_0/g$
  - Time to return to ground, $t_{final} = 2\ v_0/g$
- Write the statements necessary to calculate and print out a table of v_0, t_{max}, h_{max}, and t_{final} for values of v_0 from 1 to 10 m/s ($g = 9.807\ m/s^2$)

Trajectory Solution

```
double v0, hMax, tMax, tfinal;
const double g = 9.807;
for ( v0 = 1; v0 <= 10.5; v0++ )  {
    hMax = v0 * v0 / g;
    tMax = v0 / g;
    tFinal = 2 * v0 / g;
    cout << setw(4) << v0 << setw(9)
         << hMax << setw(9) << tMax
         << setw(9) << tFinal;
}
```


Another Trajectory Exercise

- The elevation above ground, z, for a particle shot from the ground at time = 0 with an initial velocity v_0 is given by the following equation $z = v_0 t - gt^2/2$
- This equation is valid for $0 \le t \le 2v_0/g$
- Write the C++ code to calculate and print the elevation z as a function of time t so that there are 20 steps between t = 0 (when z = 0) and $t_{max} = 2v_0/g$ for input v_0

Another Exercise Solution

```
double v0; const double g = 9.807;
cout << "Enter v0: ";
cin >> v0;
deltaTime = 2 * v0 / g / 20;
for (int i = 0; i <= 20; i++ ) {
   double t = i * deltaTime;
   double z = v0 * t - g * t * t / 2;
   cout << setw(10) << t << setw(10)
        << z;
}
```

Sentinel Exercise

- Read input data to do calculations until a "sentinel" value is read
- Sentinel is a value that will never be used for data
  - Example: a program that reads a list of data on ages from a file can exit if a negative age is entered
- Write a loop structure to read data from a file and stop when input is < 0
  - Get sum and count to compute average

California State University Northridge 19

Sentinel Solution

```
ifstream inFile( "age.dat" );
double age, sum = 0;
int n = 0;
inFile >> age;
while ( age >= 0 )
{
    sum += age;
    n++;
    inFile >> age;
}
if ( n > 0 )
    cout << "The average age is "
         << sum / n;
```

California State University Northridge 20

Another Sentinel Solution

```
ifstream inFile( "age.dat" );
double age, sum = 0; int n = 0;
do {
   inFile >> age;
   if ( age >= 0 ) {
       sum += age;
       n++;
   }
}
while ( age >= 0 )
if ( n > 0 )
    cout << "The average age is "
         << sum / n;
```

California State University Northridge 21

Looping Problems

- Midterm (April 7) and April 5 quiz will have problems like these
  - For an equation y = f(x) (e.g., $y = x^2$), print a table of x and y for a range of x values
  - For an equation z = f(x,y) (e.g., KE = $mV^2/2$), print a table of z for a range of x values and a range of y values
  - Read data from a file or keyboard and take some actions on each data item until the user enters a "sentinel" value ending input

California State University Northridge 22

Another Sample Problem

- Ask a user for input of two numbers
- Compute the sum of all even numbers in the range input by the user, including the user input values
  - E. g. if the user inputs 6 1 your program should compute 2 + 4 + 6 = 12
  - Watch out for larger number as first input and input of odd numbers as either start or finish

California State University Northridge 23

Solution

```
int first, last, i, sum = 0;
cin << first << last;
if ( first > last )
{
   int temp = first;
   first = last;
   last = temp;
}
if ( first % 2 != 0 ) first++;
for ( i = first; i <= last; i += 2 )
    sum += i;
```

- Makes sure first number in sum is even
- What about last number?

California State University Northridge 24

Question about Solution

```
if ( first % 2 != 0 ) first++;
for ( i = first; i <= last; i += 2 )
    sum += i;
```

- The first statement assures that the initial number in the sum is even
- What about the last number?
  - If last is even, the i <= last continuation condition will include it in the sum
  - If last is odd the condition will include the last even number (i <= 7 includes 6, not 8)

California State University Northridge

25