Stringstream Tutorial Using A Stringstream To Read Input From A Csv File

Rate Topic: ★★★★★ 2 Votes

r.stiltskin

D.I.C Lover

Mentor

DIC++

Reputation: 1833

Posts: 4,927
Joined: 27-December 05

#1

Posted28 March 2009 - 12:17 AM

You can think of a stringstream as a file loaded into something resembling a string, or alternatively, as a sort of string that you can write to and read from like a file. It's not exactly either of those things, but read on and it should become clearer...

A stringstream works essentially the same as an input/output file stream. You need the preprocessor directive `#include <sstream>`, declare a stringstream just like an fstream, for example
`stringstream ss;`
and, like an fstream or cout, you can write to it:
`ss << myString;` or
`ss << myCstring;` or
`ss << myInt;`, or float, or double, etc.

and you can read from it:
`ss >> myChar;` or
`ss >> myCstring;` or
`ss >> myInt;` This is also an easy way to convert strings of digits into ints, floats or doubles.

You can get the entire contents of the stringstream as a single C++ string:
`string s = ss.str();`

And it also inherits many other members from istream and ostream like get, getline, read, write, put, ... In this code I used the stringstream as an intermediate step between an input (text) file and an int array to help deal with the fact that the data within each line of the file was separated by commas, but the lines are separated only by newlines.

Following is a sample program using a stringstream to read numbers from a csv file named "input.txt" into a 6 row by 5 column int array and then prints the array.

It can take data from an input (text) file that looks like this:

and this is what ends up in the array:

```
1 3 10 3 1
1 10 10 3 10
1 0 0 0 0
0 0 0 1 10
1 1 0 2 2
10 1 0 1 10
```

It uses both forms of the istream getline member. Both of them take characters from the stream, copy them into a char array and terminate them with a '\0'.
The first one reads up to n-1 characters or until it reaches a newline or eof:

```
istream& getline (char* s, streamsize n );
```

The second one reads up to n-1 characters or until it reaches the character specified as 'delim' or eof:

```
istream& getline (char* s, streamsize n, char delim );
```

```
#include <iostream>
#include <fstream>
#include <sstream>
usingnamespacestd;

constintROWS = 6;
constintCOLS = 5;
constintBUFFSIZE = 80;

intmain() {
  intarray[ROWS][COLS];
  charbuff[BUFFSIZE];// a buffer to temporarily park the data
  ifstream infile("input.txt");
  stringstream ss;
  for(introw = 0; row < ROWS; ++row ) {
    // read a full line of input into the buffer (newline is
    //  automatically discarded)
    infile.getline( buff,  BUFFSIZE );
    // copy the entire line into the stringstream
    ss << buff;
    for(int col = 0; col < COLS; ++col ) {
      // Read from ss back into the buffer.  Here, ',' is
      //  specified as the delimiter so it reads only until
      //  it reaches a comma (which is automatically
      //  discarded) or reaches the 'eof', but of course
      //  this 'eof' is really just the end of a line of the
      //  original input.  The "6" is because I figured
      //  the input numbers would be 5 digits or less.
      ss.getline( buff, 6, ',' );
      // Next, use the stdlib atoi function to convert the
      //  input value that was just read from ss to an int,
      //  and copy it into the array.
      array[row][col] = atoi( buff );
```

The above code was set up explicitly to load data into a 6x5 array because that's what a particular application required, but it can be easily modified to read an entire input file of indeterminate length using a while loop, i.e. `while( infile.getline( buff, 50 ) )`, containing lines with any number of comma-separated values (as long as a big enough buffer is provided). Here's an example of that (it only prints the first 10 columns of the first 10 rows of the array so as not to fill your screen with 0's:

```
#include <iostream>
#include <fstream>
#include <sstream>
usingnamespacestd;

constintROWS = 100;
constintCOLS = 80;
constintBUFFSIZE = 80;

intmain() {
  // The array must be big enough to fit the input data.
  intarray[ROWS][COLS] = {0};
  introw, col;
  charbuff[BUFFSIZE];// a buffer to temporarily park the data
  ifstream infile("input.txt");
  stringstream ss;
  // Read input into the buffer a line at a time, until
  //  end of file is reached (newlines are automatically
  //  discarded).  The buffer must be big enough to fit
  //  an entire line from the file.
  // Notice that while reading from the file we check how
  //  many rows have been read, to avoid writing beyond
  //  the end of the array.
  row = 0;
  while( infile.getline( buff,  BUFFSIZE ) && row < ROWS ) {
    // copy the entire buffered line into the stringstream
    ss << buff;
    // Read from ss back into the buffer.  Now, ',' is
    //  specified as the delimiter so it reads only until
    //  it reaches a comma (which is automatically
    //  discarded) or reaches the 'eof', but of course
    //  this 'eof' is really just the end of a line of the
    //  original input.  The "10" means this will handle
```

This post has been edited by**r.stiltskin**: 02 April 2009 - 10:42 PM

Is This A Good Question/Topic? 3

MultiQ