

# Programming with cin, cout, and Data Files

July 13, 2010

# Object Oriented Programming

**class** A class is a mechanism that allows a programmer to define new data types.

- ▶ A class can be used to add functionality to an existing data type or to create a new data type.
- ▶ A class definition combines data and functionality.

**object** An object is a variable of a defined class type, also referred to as an **instance** of a class .

## cin & cout as Instances

cin is an input stream

- ▶ cin object of type istream
- ▶ istream class is defined in file iostream
  - ▶ #include <iostream>

```
1 int N(0); // number of datapoints
2 double sumx(0), sumy(0);
3 double x, y;
4 while( cin >> x >> y ) {
5     sumx += x;
6     sumy += y;
7     N++;
8 }
9 if( N ) {
10    cout.setf( ios::fixed );
11    cout.precision( 3 );
12    cout << "Averages_";
13    cout << "_x:" << sumx/N;
14    cout << "_y:" << sumy/N;
15 }
```

## cin & cout as Instances

cout is an output stream

- ▶ cout object of type ostream
- ▶ ostream class is defined in file iostream
  - ▶ #include <iostream>
- ▶ **Member functions!**
  - ▶ cout.setf( ios::fixed );
  - ▶ cout.precision(3);

```
1 int N(0); // number of datapoints
2 double sumx(0), sumy(0);
3 double x, y;
4 while( cin >> x >> y ) {
5     sumx += x;
6     sumy += y;
7     N++;
8 }
9 if( N ) {
10    cout.setf( ios::fixed );
11    cout.precision( 3 );
12    cout << "Averages_";
13    cout << "_x:" << sumx/N;
14    cout << "_y:" << sumy/N;
15 }
```

Try questions 2 and 5.

## Goals

- ▶ Learn how to reliably read input from `cin`.
- ▶ Leverage our familiarity of `cout` and `cin` to write and read from our own files (not just the keyboard and console).
- ▶ Learn how to format the output of `cout`.

## cin Properties

- ▶ cin ignores all *leading* whitespace or line endings before it parses “real” (non-whitespace) character data.
- ▶ once the first valid character of data has been read, cin *stops* parsing ...
  - ▶ **successfully** at whitespace or end-of-file.
  - ▶ **successfully** at the first nonsensical character for the type of value being read.
- ▶ if at least one valid character of data **can not be read**, then cin enters the **failure** state!
- ▶ “nonsensical” depends on *where* the character is encountered.  
Only the fourth plus symbol is nonsensical in this stream of integers:

+1234 +123++4

- ▶ Demo:

[Edit Program](#)

[Run Program](#)

## Reading Whole Lines

Why can't we read whole lines in C++?

## Reading Whole Lines

Why can't we read whole lines in C++?

You can, by reading into a string variable type with the `getline` function:

```

1  /**
2   * Display numbered lines of cin input
3   */
4  string inputLine;
5  int lineNo(1);
6
7  cout << "Enter lines of any length." ;
8  cout << "CTRL-D or CTRL-Z to quit" << endl;
9
10 // The '\n' represents the newline character,
11 // otherwise known as the Enter key.
12 while( getline( cin, inputLine, '\n' ) ) {
13     cout << lineNo++ << ":\n";
14     cout << inputLine << endl;
15 }

```

<<Interactive Program>>

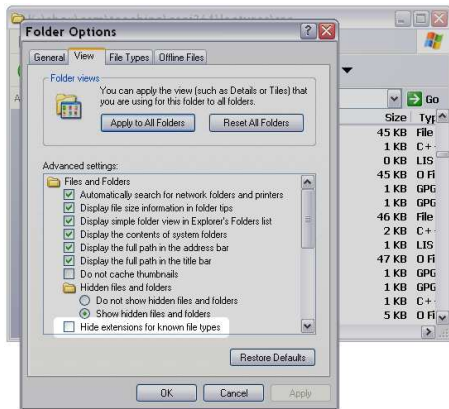
RUN

EDIT

../src/O\_t.cxx

# File I/O

## Some Disk File Details



1. An “open file” in C++ **does not** mean a new window will appear on your screen.
2. Don't use spaces or punctuation when naming files (name them like valid variable names).
3. Windows, by default, **does not** show you the entire filename, it typically hides the file “extension” (.txt, .cpp, ...).



## Simple File Pattern: Reading Data

```
1 ifstream inFile( "data3.dat" );
2 if( !inFile ) {
3     cout << "Error opening file." << endl;
4     exit(1);
5 }
6 cout << "File opened!" << endl;
7
8 // Now read data from file. Print every other
9 // number (beginning with the first)
10 int counter(0);
11 double data;
12 while( inFile >> data ) {
13     if( ++counter % 2 ) {
14         cout << counter << " " << data
15             << endl;
16     }
17 }
18
19 cout << "Total " << counter << " data points."
20     << endl;
21
22 inFile.close();
```

### Run Read Example

- ▶ Simple, not very flexible, but it works well.
- ▶ Opens file on variable declaration.
- ▶ Note the !inFile to check failure.
- ▶ **NICE!** The while loop reads from the file until all data is consumed, or there is a parsing error.

Engineers need to do this a lot!

# Simple File Pattern: Writing Data

## Run Write Example

- ▶ Simple, not very flexible, but it works well.
- ▶ Opens (**overwrites!**) file on variable declaration.
- ▶ Note the `!outFile` to check failure.

```
1 ofstream outFile( "JUNK~" );
2 if( !outFile ) {
3     cout << "Error opening file." << endl;
4     exit(1);
5 }
6
7 // write some data
8 outFile << "First_line_in_JUNK~" << endl;
9 // a bool, a char, an int, and a double
10 char letter( 'K' );
11 double avogadro(6.0221415 e23);
12 outFile << true << "\t" << letter << "\t" <<
13     104729 << "\n" << avogadro << endl;
14
15 // close the file, always!
16 outFile.close();
```

## Simple File Pattern Similarities

- ▶ `ifstream` and `ofstream` types are both from `#include <fstream>`
- ▶ Test open success with `if( !fileVariable ) ...`
- ▶ `exit(1)` on failure (in `#include <cstdlib>`)
- ▶ `.close()` when finished with file.

# Reading Data

# Simple Average

## Averages from cin

```

1 int N(0); // number of datapoints
2 double sumx(0), sumy(0);
3 double x, y;
4 while( cin >> x >> y ) {
5     sumx += x;
6     sumy += y;
7     N++;
8 }
9 if( N ) {
10     cout.setf( ios::fixed );
11     cout.precision( 3 );
12     cout << "Averages_";
13     cout << "_x:" << sumx/N;
14     cout << "_y:" << sumy/N;
15 }

```

## Run Program

## Data File

```

1 ifstream infile( "data3.dat" );
2 if( !infile ) {
3     cout << "Error_opening_file." << endl;
4     exit(1);
5 }
6
7 int N(0); // number of datapoints
8 double sumx(0), sumy(0);
9 double x, y;
10 while( infile >> x >> y ) {
11     sumx += x;
12     sumy += y;
13     N++;
14 }
15 if( N ) {
16     cout.setf( ios::fixed );
17     cout.precision( 3 );
18     cout << "Averages";
19     cout << "_x:" << sumx/N;
20     cout << "_y:" << sumy/N;
21 }
22
23 infile.close();

```

## inFile » strData vs. getline

```

1 ifstream inFile( "asciibeast.txt" );
2 if( !inFile ) {
3     cout << "asciibeast.txt_could_not_be_opened." << endl;
4     exit(1);
5 }
6
7 string data;
8 while( getline( inFile, data, '\n' ) ) {
9     cout << data << endl;
10 }
11
12 inFile.close();

```

Run  
getline

Run  
inFile >> strData

```

1 ifstream inFile( "asciibeast.txt" );
2 if( !inFile ) {
3     cout << "asciibeast.txt_could_not_be_opened." << endl;
4     exit(1);
5 }
6
7 string data;
8 while( inFile >> data ) {
9     cout << data;
10 }
11
12 inFile.close();

```

## » Equations

## Understanding » Equations

What steps are taken for the calculation of R?

```
1 int i1 (1), i2 (2), i3 (3);  
2 double d1 (10.5), R;  
3  
4 R = i1 + i2 + d1 + i3;
```

## Understanding » Equations

```
1 int i1(1), i2(2), i3(3);
2 double d1(10.5), R;
3
4 R = i1 + i2 + d1 + i3;
5 R = tmpInt1 + d1 + i3; // i1 + i2 -> tmpInt1
6 R = tmpDb11 + i3; // tmpInt1 + d1 -> tmpDb11
7 R = tmpDb12; // tmpDb11 + i3 -> tmpDb12
8 // tmpDb12 saved into R
```

The compiler simplification of this equation is analogous to the simplification of algebraic expressions in mathematics.

## Understanding » Equations

How is this equation similar or different than the last?

```
1 int i1 , i2 , i3 ;  
2 double d1 ;  
3  
4 cin >> i1 >> i2 >> d1 >> i3 ;
```

## Understanding » Equations

```

1 int i1 , i2 , i3 ;
2 double d1 ;
3
4 cin >> i1 >> i2 >> d1 >> i3 ;
5     cin >> i2 >> d1 >> i3 ; // cin >> i1 → cin + s/e
6         cin >> d1 >> i3 ; // cin >> i2 → cin + s/e
7             cin >> i3 ; // cin >> d1 → cin + s/e
8                 cin ; // cin >> i3 → cin + s/e

```

Again, the simplification is similar to that of mathematics, as long as you understand that:

`cin >> variable` → `cin` + **side effect**

## Understanding » Equations

```
1 int i1 , i2 , i3 ;
2 double d1 ;
3
4 cin >> i1 >> i2 >> d1 >> i3 ;
5     cin >> i2 >> d1 >> i3 ; // cin>>i1 -> cin + s/e
6         cin >> d1 >> i3 ; // cin>>i2 -> cin + s/e
7             cin >> i3 ; // cin>>d1 -> cin + s/e
8                 cin ; // cin>>i3 -> cin + s/e
9                     // ... cin "thrown-away"
```

There was no assignment operator (=, +=, ...) in the expression, so the final value (cin) is “thrown away” by the compiler.

Try question 7.

## Stream Conversions to Boolean

In C++,  $0 == \text{false}$ , any other number is true (**We know this already**).

## Stream Conversions to Boolean

In C++, 0==false, any other number is true (**We know this already**).

**New Stuff**  $\implies$  C++ converts any stream (cin or cout, or ifstream, ofstream) to a Boolean that tells if an input or output error has occurred.

**No I/O Error**  $\rightarrow$  true

## Stream Conversions to Boolean

In C++, 0==false, any other number is true (**We know this already**).

**New Stuff**  $\implies$  C++ converts any stream (cin or cout, or ifstream, ofstream) to a Boolean that tells if an input or output error has occurred.

**No I/O Error**  $\rightarrow$  true

```

1 int N(0); // number of datapoints
2 double sumx(0), sumy(0);
3 double x, y;
4 while( cin >> x >> y ) {
5     sumx += x;
6     sumy += y;
7     N++;
8 }
9 if( N ) {
10    cout.setf( ios::fixed );
11    cout.precision( 3 );
12    cout << "Averages_";
13    cout << "_x:" << sumx/N;
14    cout << "_y:" << sumy/N;
15 }

```

When a parsing error or end-of-file is encountered, the while() loop is broken.

**Why?**

## The Reading Data Mantra

**DON'T** use member functions to detect failures  
(**avoid** `.eof()` and `.fail()`),

**DO USE** one-liner input statements as predicates,

**DO USE** stream conversion to Boolean to detect failures.

## Formatting cout

Operation	C++ statement
Print left justified	<code>cout.setf( ios::left );</code>
Print right justified	<code>cout.setf( ios::right );</code>
Print in a fixed width	<code>cout.width(10);</code>
( <b>Optionally</b> ) use fixed decimal points	<code>cout.setf( ios::fixed );</code>
Specify fixed points	<code>cout.precision(4);</code>
( <b>Optionally</b> ) use scientific notation	<code>cout.setf( ios::scientific );</code>
Print words for Booleans	<code>cout.setf( ios::boolalpha );</code>

See textbook § 6.2 for details. You may need to call `cout.unsetf()` to “unset” left or right justification.

# Formatting cout

## Data File

```

7 // console cell numbers
8 for( int i(1); i<=40; i++ ) cout << i % 10;
9 cout << endl;
10
11 // reformat sets of four data elements into
12 // horizontal 'records '.
13 int row(1), x;
14 double y, z;
15 string s;
16 while( inputfile >> x >> y >> z >> s ) {
17     cout.unsetf( ios::right ); // undo
18     cout.setf( ios::left ); // left justify
19     cout.width( 4 ); cout << row++ << " ";
20     cout.width( 4 ); cout << x << " ";
21     cout.setf( ios::fixed );
22     cout.precision( 3 );
23     cout.width( 9 ); cout << y << "|";
24     cout.unsetf( ios::left ); // undo
25     cout.setf( ios::right ); // right justify
26     cout.setf( ios::scientific );
27     cout.precision( 5 );
28     cout.width(14); cout << z << " ";
29     cout.width(5); cout << s << endl;
30 }
31
32 // another set of cell numbers
33 for( int i(1); i<=40; i++ ) cout << i % 10;
34 cout << endl;

```

```

1201 0.001
314159002718 a
203 0.0001
30123483751 ab
3748 0.00001
31231298.4320 c
837 1e-6
.312323198774 abc
38 0.002
83.7493343890 abc

```

## Example Output

```

1234567890123456789012345678901234567890
1    1201 0.001    |    3.1416e+11    a
2    203 0.0001   |    3.0123e+10    ab
3    3748 1e-05   |    3.1231e+07    c
4    837 1e-06   |    0.31232    abc
5    38 0.002    |    83.749    abc
1234567890123456789012345678901234567890

```

## Formatting cout

- ▶ `.precision()`,  
`.setf( ios::scientific )`,  
and `.setf( ios::fixed )`,  
appear to be mere suggestions to  
the formatting engine in C++.  
**Not absolute rules.**
- ▶ Significant digits appear to be  
highly respected.

### Data File

```
1201 0.001
314159002718 a
203 0.0001
30123483751 ab
3748 0.00001
31231298.4320 c
837 1e-6
.312323198774 abc
38 0.002
83.7493343890 abc
```

### Example Output

```
1234567890123456789012345678901234567890
1    1201 0.001   |    3.1416e+11   a
2    203 0.0001  |    3.0123e+10   ab
3    3748 1e-05  |    3.1231e+07   c
4    837 1e-06   |    0.31232      abc
5    38  0.002   |    83.749       abc
1234567890123456789012345678901234567890
```

# Formatting cout

```

7 // console cell numbers
8 for( int i(1); i<=40; i++ ) cout << i % 10;
9 cout << endl;
10
11 // reformat sets of four data elements into
12 // horizontal 'records '.
13 int row(1), x;
14 double y, z;
15 string s;
16 while( inputfile >> x >> y >> z >> s ) {
17     cout.unsetf( ios::right ); // undo
18     cout.setf( ios::left ); // left justify
19     cout.width( 4 ); cout << row++ << " ";
20     cout.width( 4 ); cout << x << " ";
21     cout.setf( ios::fixed );
22     cout.precision( 3 );
23     cout.width( 9 ); cout << y << "|";
24     cout.unsetf( ios::left ); // undo
25     cout.setf( ios::right ); // right justify
26     cout.setf( ios::scientific );
27     cout.precision( 5 );
28     cout.width(14); cout << z << " ";
29     cout.width(5); cout << s << endl;
30 }
31
32 // another set of cell numbers
33 for( int i(1); i<=40; i++ ) cout << i % 10;
34 cout << endl;

```

## Bad Data File

```

1201 0.001
314159002718 abcdefg
203 0.0001
30123483751 ab
3748 0.00001
31231298.4320 c
8374382 1e-6
.312323198774 abc
38 0.002
83.7493343890 abc

```

## Example Output

```

1234567890123456789012345678901234567890
1 1201 0.001 | 3.1416e+11 abcdefg
2 203 0.0001 | 3.0123e+10 ab
3 3748 1e-05 | 3.1231e+07 c
4 8374382 1e-06 | 0.31232 abc
5 38 0.002 | 83.749 abc
1234567890123456789012345678901234567890

```

## Formatting cout

When values don't conform to the expected output windows, C++ considers the data **more important** than the formatting.

### Bad Data File

```
1201 0.001
314159002718 abcdefg
203 0.0001
30123483751 ab
3748 0.00001
31231298.4320 c
8374382 1e-6
.312323198774 abc
38 0.002
83.7493343890 abc
```

### Example Output

```
1234567890123456789012345678901234567890
1 1201 0.001 | 3.1416e+11 abcdefg
2 203 0.0001 | 3.0123e+10 ab
3 3748 1e-05 | 3.1231e+07 c
4 8374382 1e-06 | 0.31232 abc
5 38 0.002 | 83.749 abc
1234567890123456789012345678901234567890
```

Complete questions 9 and 12.

**finis**  
(Reference Slides Follow)

## Reference: Using an Input File from Disk

<b>Required</b> Preprocessor	<code>#include &lt;fstream&gt;</code>
Declare & Open	<code>ifstream inFile( "file.dat" );</code>
Open name in string fname	<code>ifstream inFile;</code> <code>inFile.open( fname.c_str() );</code>
Read values from file	<code>inFile » carltime » carlaccel;</code>
$\infty$ Value Reading	<code>while( inFile » var ) { ... }</code>
Return read position to “home”	<code>inFile.clear(), inFile.seekg(0);</code>
close File	<code>inFile.close();</code>
File OK?	<code>if( !inFile ) { ... }</code>
“Reset” File to OK	<code>inFile.clear();</code>

## Reference: Using an Output File on Disk

<b>Required</b> Preprocessor	<code>#include &lt;fstream&gt;</code>
Open and Overwrite "file.dat"	<code>ofstream outFile("file.dat");</code>
Open and Append "file.dat"	<code>ofstream outFile("file.dat", ios::app);</code>
Write to name in string fname	<code>ofstream outFile; outFile.open( fname.c_str() );</code>
Write constants or variables (diameter) to file	<code>outFile &lt;&lt; 3.14 &lt;&lt; diameter &lt;&lt; endl;</code>
close File	<code>outFile.close();</code>
File OK?	<code>if( !outFile ) { ... }</code>
“Reset” File to OK	<code>outFile.clear();</code>