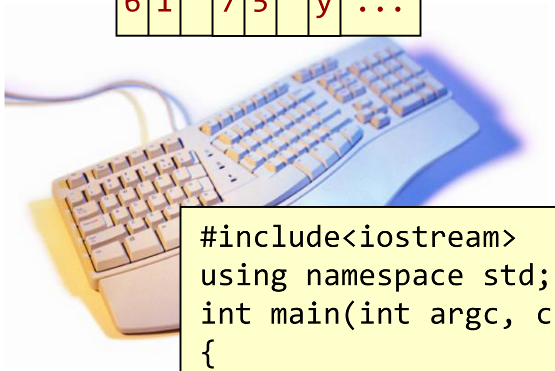
CS 103 Unit 14 – Stringstreams and Parsing

I/O Streams

- '>>' operator used to read data from an input stream
  - Always stops at whitespace
- '<<' operator used to write data to an output stream
  - 'endl' forces a flush…Flush forces the OS to move data from the internal OS stream to the actual output device (like the monitor)

input stream (user types all at once):

6	1		7	5		y	...



```
#include<iostream>
using namespace std;
int main(int argc, char *argv[])
{
  int dummy, x;
  cin >> dummy >> x;

}
```

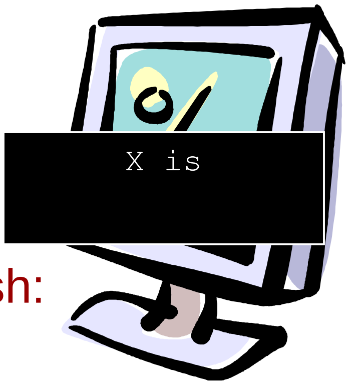
input stream:

	y	...

```
#include<iostream>
using namespace std;
int main(int argc, char *argv[])
{
  cout << "X is " << endl;
  cout << 4;
}
```

output stream in OS:

I	t		w	a	s		t	h	e	\n	4



output stream after flush:

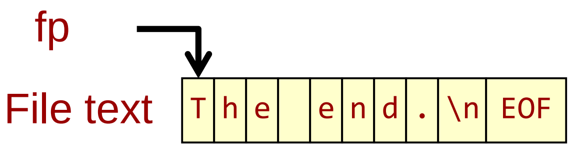
4

Kinds of Streams

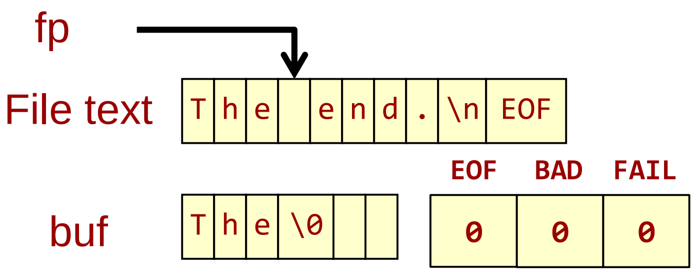
- I/O streams
  - Keyboard (cin) and monitor (cout)
- File streams – Contents of file are the stream of data
  - #include <fstream> and #include <iostream>
  - ifstream and ofstream objects

When Does It Fail

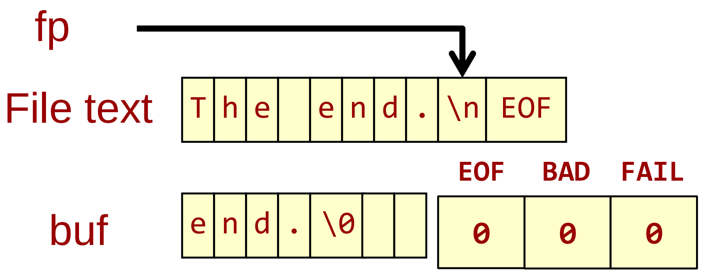
```
char buf[40];
ifstream inf(argv[1]);
```



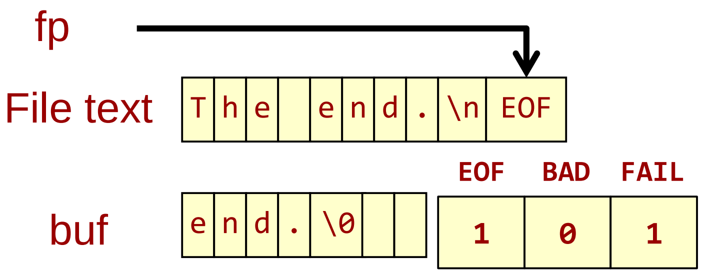
```
inf >> buf;
```



```
inf >> buf;
```



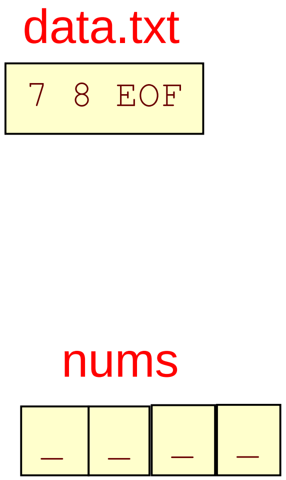
```
inf >> buf;
```



For filestreams & now stringstreams the stream doesn't fail until you read ***PAST*** the EOF. Reading something that stops ON the EOF will not cause fail() to return true

Which Option Works?

```
#include<iostream>
#include<fstream>
using namespace std;
int main()
{
 vector<int> nums;
 ifstream ifile("data.txt");
 int x;
 while( !ifile.fail() ){
   ifile >> x;
   nums.push_back(x);
 }
 ...
}
```



```
#include<iostream>
#include<fstream>
using namespace std;
int main()
{
 vector<int> nums;
 ifstream ifile("data.txt");
 int x;
 while( 1 ){
   ifile >> x;
   if(ifile.fail()) break;
   nums.push_back(x);
 }
 ...
}
```

Goal is to read all integers from the file into a vector. Which of the 3 works?

A More Compact Way

```
#include<iostream>
#include<fstream>
using namespace std;
int main()
{
 vector<int> nums;
 ifstream ifile("data.txt");
 int x;
 while( !ifile.fail() ){
   ifile >> x;
   nums.push_back(x);
 }
 ...
}
```

✗

data.txt

```
7 8 EOF
```

nums

_	_	_	_

```
#include<iostream>
#include<fstream>
using namespace std;
int main()
{
 vector<int> nums;
 ifstream ifile("data.txt");
 int x;
 while( 1 ){
   ifile >> x;
   if(ifile.fail()) break;
   nums.push_back(x);
 }
 ...
}
```

✓

```
int x;
while( ifile >> x ){
  nums.push_back(x);
}
...
```

✓

Calling >> on an input stream will essentially return a Boolean:

- true = success
- false = failure

Correct Pattern for File I/O or Streams

- Step 1: **Try** to read data (>> or getline)
- Step 2: **Check** if you failed
- Step 3: Only **use** the data read from step 1 if you succeeded

- If you read and then use the data BEFORE checking for failure, you will likely get 1 extra (bogus) data value at the end

Recall How To Get Lines of Text

- Using the >> operator to get an input string of text (char * or char [] variable passed to cin) ***implicitly stops at the first whitespace***
- How can we get a whole line of text (including spaces)
  - **cin.getline(char *buf, int bufsize);**
  - **ifile.getline(char *buf, int bufsize);**
  - Reads max of bufsize-1 characters (including newline)
- But **getline()** uses **char* (C-Strings)**… what if we want to use C++ strings???

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

int main ()

{

  char myline[100]; int i = 1;

  ifstream ifile ("input.txt");

  if(  ifile.fail() ){  // can't open?
    return 1;
  }

  ifile.getline(myline, 100);
  while ( ! ifile.fail()) {
   cout << i++ << ": " << myline << endl;
   ifile.getline(myline, 100);
  }

  ifile.close();
  return 0;
}
```

input.txt

```
The fox jumped over the log.

The bear ate some honey.

The CS student solved a hard problem.
```

```
1: The fox jumped over the log.

2: The bear ate some honey.

3: The CS student solved a hard problem.
```

C++ String getline()

- C++ string library (#include <string> defines a global function (not a member of ifstream or cin) that can read a line of text into a C++ string
- Prototype: **istream& getline(istream &is, string &str, char delim);**
  - `is` = any input stream (ifstream, cin), etc.)
  - `str` = A C++ string that it will fill in with text
  - `delim` = A char to stop on (by default it is '\n') which is why its called getline
  - Returns the updated istream (the 'is' object you passed in as the 1st arg)
- The text from the input stream will be read up through the first occurrence of 'delim' (defaults to '\n') and placed into str. The delimiter will be stripped from the end of str and the input stream will be pointing at the first character after 'delim'.

```
ifstream myfile(argv[1]);
string myline;
// Not a member function
myfile.getline( myline );  // doesn't work

// global scope function...correct
getline(myfile, myline);
```

```
int line_no = 0;
ifstream myfile(argv[1]);
string myline;
getline( myfile, myline );
cout << myline << endl;

while ( getline( myfile, myline ) ) {
  cout << "Line: " << myline << endl;
}
```

STRINGSTREAMS

Introducing…Stringstreams

- I/O streams
  - Keyboard (`cin`) and monitor (`cout`)
- File streams – Contents of file are the stream of data
  - #include <fstream> and #include <iostream>
  - `ifstream` and `ofstream` objects
- Stringstreams – Contents of a string are the stream of data
  - #include <sstream> and #include <iostream>
  - `sstream` object

C++ String Stream

- If streams are just sequences of characters, aren't strings themselves like a stream?
  - The <sstream> library lets you treat C++ string objects like they were streams
- Why would you want to treat a string as a stream?
  - Buffer up output for later display
  - Parse out the pieces of a string
  - Data type conversions
- Very useful in conjunction with string's getline(...)

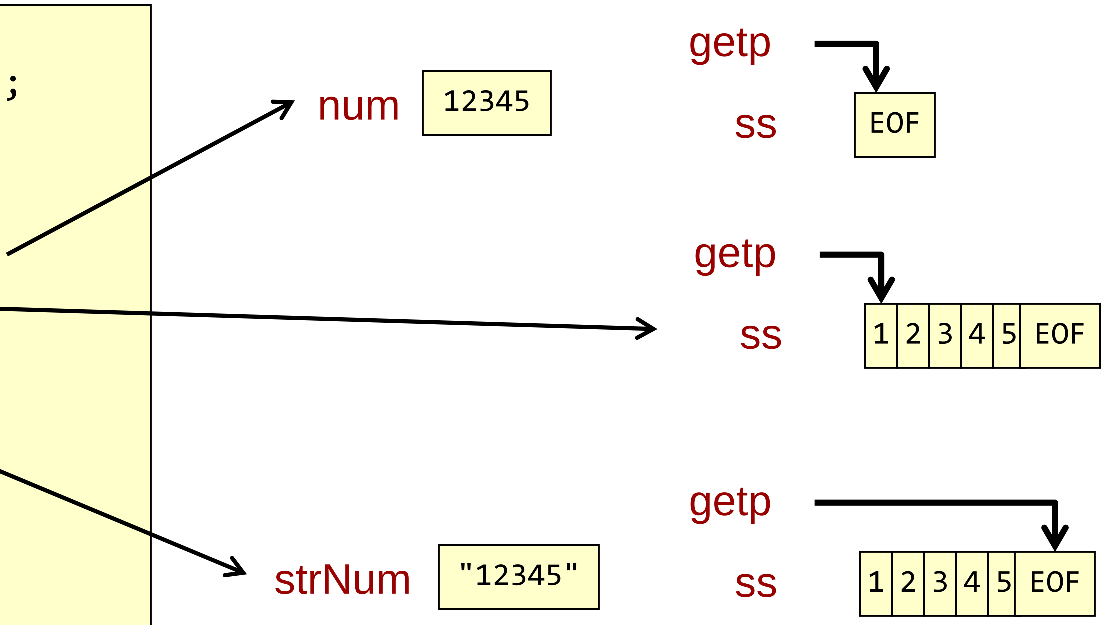
C++ Stringstream: Application 1a

- Use << and >> to convert numbers into strings (i.e. 12345 => "12345")

```
#include<sstream>
using namespace std;
int main()
{
 stringstream ss;
 int num = 12345;
 ss << num;

 string strNum;
 ss >> strNum;

 return 0;
}
```



sstream_test1.cpp

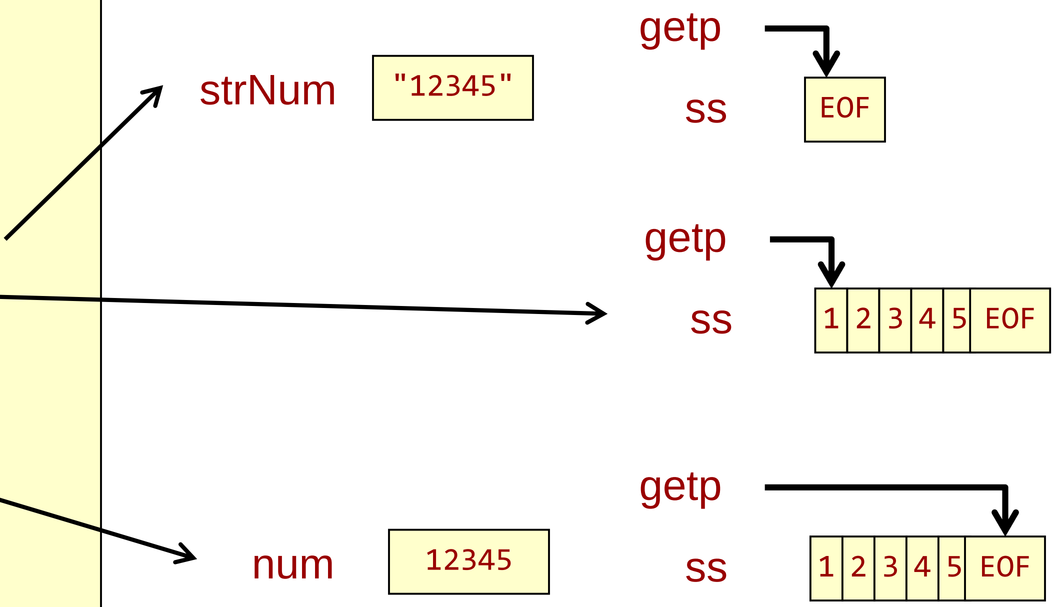
C++ Stringstream: Application 1b

- Use << and >> to convert strings into numbers (i.e. "12345" => 12345)

```
#include<sstream>
using namespace std;
int main()
{
 stringstream ss;
 string strNum = "12345";
 ss << strNum;

 int num;
 ss >> num;
 return 0;
}
```

sstream_test2.cpp



C++ Stringstream: Application 2

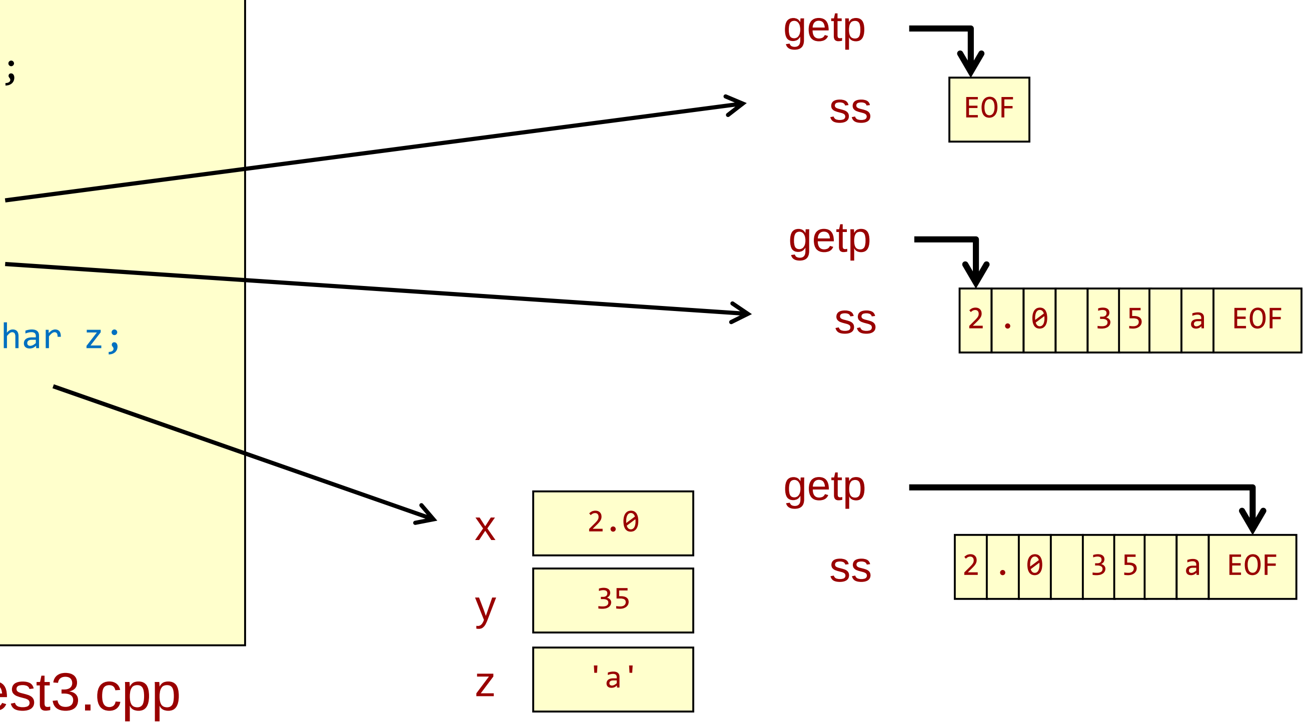
- Can parse (split) a string of many values into separate variables

```
#include <sstream>
using namespace std;
int main()
{
 stringstream ss;
 ss << "2.0 35 a";

 double x, int y; char z;
 ss >> x >> y >> z;

 return 0;
}
```

sstream_test3.cpp



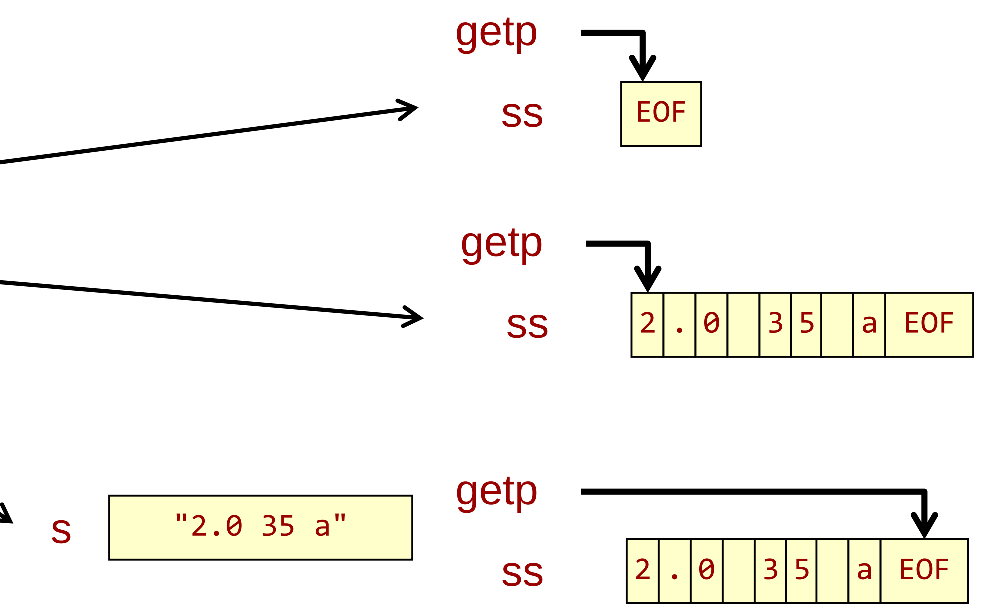
C++ Stringstream: Application 3

- Use the `.str()` member function to create a large string from many value (i.e. return a string with the contents of whatever is in the stream)

```
#include<sstream>
using namespace std;
int main()
{
 stringstream ss;
 ss << 2.0 << " " << 35;
 ss << " " << 'a';

 string s = ss.str();

 return 0;
}
```



sstream_test4.cpp

C++ Stringstream Reuse

- Beware of re-using the same stringstream object for multiple conversions. It can be weird.
  - Make sure you clear it out between uses and re-init with an empty string
- Or just make a new stringstream each time

```
stringstream ss;

//do something with ss

ss.clear();
ss.str("");
// now you can reuse ss
```

Option 1: Reuse

```
stringstream ss;

//do something with ss


// Just declare another stream
stringstream ss2;
// do something with ss2
```

Option 2: Use new stringstream

Exercise

- What's in each variable after execution?
  - text
  - num
  - val

```
string text;
int num;
double val;

stringstream ss("Hello 103 2.0");
ss >> text >> num >> val;
```

Exercises

- In class exercises
  - Stringstream_in
  - Stringstream_out
  - Date

Choices

Where is my data?

Keyboard (use ______)

File (use ______)

String (use _______)

Do I know how many?

Yes

No

Choices

Is it delimited?

Yes

No

What type of data?

Text

Integers/ Doubles

Choosing an I/O Strategy

- Is my data delimited by particular characters?
  - Yes, stop on newlines: Use getline()
  - Yes, stop on other character: User getline() with optional 3rd character
  - No, Use >> to skip all whitespaces and convert to a different data type (int, double, etc.)
- If "yes" above, do I need to break data into smaller pieces (vs. just wanting one large string)
  - Yes, create a stringstream and extract using >>
  - No, just keep the string returned by getline()
- Is the number of items you need to read known as a constant or a variable read in earlier?
  - Yes, Use a loop and extract (>>) values placing them in array or vector
  - No, Loop while extraction doesn't fail placing them in vector

Remember: getline() always gives text/string.
To convert to other types it is easiest to use >>

In-Class Exercises

- Wordcount

HELPFUL PA5 APPROACH

getline() and stringstreams

- Imagine a file has a certain format where you know related data is on a single line of text but aren't sure how many data items will be on that line
- Can we use >>?
  - No it doesn't differentiate between different whitespace (i.e. a ' ' and a '\n' look the same to >> and it will skip over them)
- We can use getline() to get the whole line, then a stringstream with >> to parse out the pieces

```
int num_lines = 0;
int total_words = 0;

ifstream myfile(argv[1]);

string myline;
while( getline(myfile, myline) ){

   stringstream ss(myline);

   string word;
   while( ss >> word )
     {  total_words++; }
   num_lines++;
}

double avg =
   (double) total_words / num_lines;

cout << "Avg. words per line: ";
cout << avg << endl;
```

```
The fox jumped over the log.

The bear ate some honey.

The CS student solved a hard problem.
```

Using Delimeters

- Imagine a file has a certain format where you know related data is on a single line of text but aren't sure how many data items will be on that line
- Can we use >>?
  - No it doesn't differentiate between different whitespace (i.e. a ' ' and a '\n' look the same to >> and it will skip over them)
- We can use getline() to get the whole line, then a stringstream with >> to parse out the pieces

Text file:

```
garbage stuff (words I care about) junk
```

```
vector<string> mywords;

ifstream myfile(argv[1]);

string myline;
getline(myfile, myline, '(');
// gets "garbage stuff "
// and throws away '('

getline(myfile, myline, ')' );
// gets "words I care about"
// and throws away ')'`

stringstream ss(myline);
string word;
while( ss >> word ) {
  mywords.push_back(word);
}
```

	0	1	2	3
mywords	"words"	"I"	"care"	"about"

SOLUTIONS

Which Option?

```
#include<iostream>
#include<fstream>
using namespace std;
int main()
{
 vector<int> nums;
 ifstream ifile("data.txt");
 int x;
 while( !ifile.fail() ){
   ifile >> x;
   nums.push_back(x);
 }
 ...
}
```

✗

data.txt

```
7 8 EOF
```

nums

_	_	_	_

```
#include<iostream>
#include<fstream>
using namespace std;
int main()
{
 vector<int> nums;
 ifstream ifile("data.txt");
 int x;
 while( 1 ){
   ifile >> x;
   if(ifile.fail()) break;
   nums.push_back(x);
 }
 ...
}
```

✓

Need to check for failure after you extract but before you store/use

```
int x;
while( ifile >> x ){
  nums.push_back(x);
}
...
```

✓

A stream returns itself after extraction
A stream can be used as a bool (returns true if it hasn't failed)