We are 183

L12: Week 8 - Monday

Reminders

- Done with Exam 1!

- Project 3 due Friday
  - First partner project

Upcoming Film Screening

LEAN IN AT THE UNIVERSITY OF MICHIGAN

presents

OFFICIAL SELECTION 2015 TRIBECA FILM FESTIVAL

CODE

DEBUGGING THE GENDER GAP

followed by facilitated dialogue

WEDNESDAY, FEBRUARY 24TH

7 PM | FREE ADMISSION!

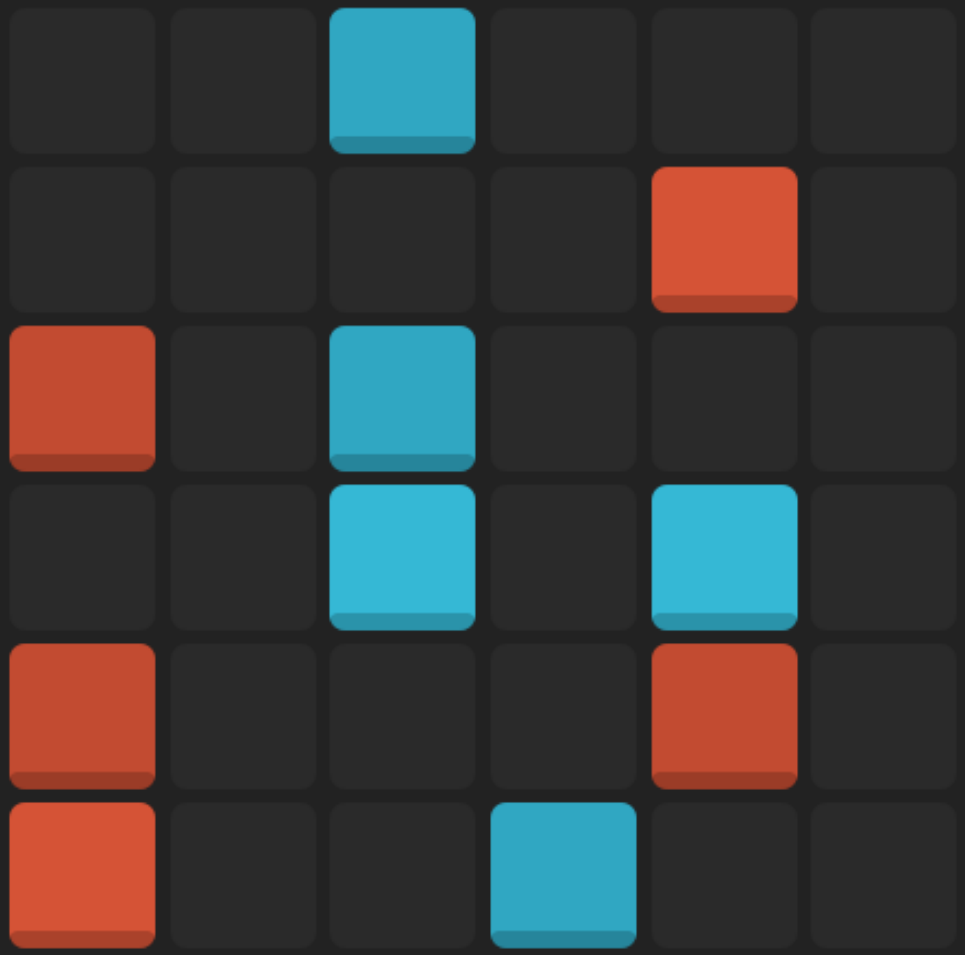
NORTH QUAD SPACE 2435

FUNDED BY

NORTH QUAD PROGRAMMING

UNIVERSITY OF MICHIGAN

6 x 6



Last Time… on EECS 183

Passing Arrays to Functions
2D Arrays

Passing Arrays to Functions

- Write a function that accepts an array of integers and squares all the elements of the array.
- Prototype:

```
void squareArray(int arr[], int size);
```

Always passed by reference.
No **&** needed

Passing Arrays to Functions

- Write a function that accepts an array of integers and squares all the elements of the array.
- Prototype:

```
void squareArray(int arr[], int size);
```

Pass the size so you won't go out-of-range.

Passing Arrays to Functions

- Write a function that accepts an array of integers and squares all the elements of the array.
- Prototype:

```
void squareArray(int arr[], int size);
```

Array modified directly.
Nothing to return.

Passing Arrays to Functions

```
void squareArray(int arr[], int size);

    const int MAX_SIZE = 5;
    int array[MAX_SIZE] = { };
    int size = 3;

    squareArray(array, size);
    squareArray(array, MAX_SIZE);
    squareArray(array, 5);
    squareArray(array, 3);
```

Passing Arrays to Functions

```
void squareArray(int arr[], int size);

    const int MAX_SIZE = 5;
    int array[MAX_SIZE] = { };
    int size = 3;

    squareArray(array, size);
    squareArray(array, MAX_SIZE);
    squareArray(array, 5);
    squareArray(array, 3);
```

Passing Arrays to Functions

```
void squareArray(int arr[][5], int row,
                 int col);

const int MAX_ROW = 5;
const int MAX_COL = 5;

int array[MAX_ROW][MAX_COL] = { };
int row_size = 3;
int col_size = 3;

squareArray(array, row_size, col_size);
squareArray(array, MAX_ROW, MAX_COL);
```

Passing Arrays to Functions

- Write a function that accepts an array of integers and squares all the elements of the array.

```
int[] squareArray(int arr[], int size);
```

Compile Error – Cannot return an array

Passing Arrays to Functions

C++ arrays are:

- **Always** passed by **reference**
  - No **&** needed

- Saves memory space
  - array is not copied

- Saves processing time
  - array elements are not copied

Prevent function from modifying an array – Similar to pass-by-value

```
// allows function to alter data
// all arrays are passed by reference
void printResults(int data[ ], int size);


// With const, function can't alter array
void printResults(const int data[ ], int size);
```

Return from a function

```
const int SIZE = 5;
void squareArray(int arr[], int size);

int main() {
    int arr1[SIZE];
    squareArray(arr1, SIZE);
    // More code…
}
```

Passed by **reference**
no need to "return" the array

Calls & Prototypes

Example Call (assume main is caller):

```
squareArray(data, SIZE);
```

Possible Corresponding Prototypes:

```
void squareArray(int data[SIZE], int size);
void squareArray(int data[ ], int size);
void squareArray(int * data, int size);
```

i>Clicker #1

Given the function declaration

```
void printElement(int array[], int index);
```

and the variable declarations

```
int array[] =  { 1, 2, 3 };
int index = 1;
```

What is the correct way to **<u>call</u>** `printElement()`?

```
A.  printElement(array[index], index);
B.  printElement(array[], index);
C.  printElement(array, index);
D.  printElement(array[index]);
```

i>Clicker #1

Given the function declaration

```
void printElement(int array[], int index);
```

and the variable declarations

```
int array[] =  { 1, 2, 3 };
int index = 1;
```

What is a correct way to **<u>call</u>** `printElement()`?

A. `printElement(array[index], index);`
B. `printElement(array[], index);`
C. `printElement(array, index);`
D. `printElement(array[index]);`

Two Dimensional Arrays

#columns

```
int arr[4][3];

...

...
```

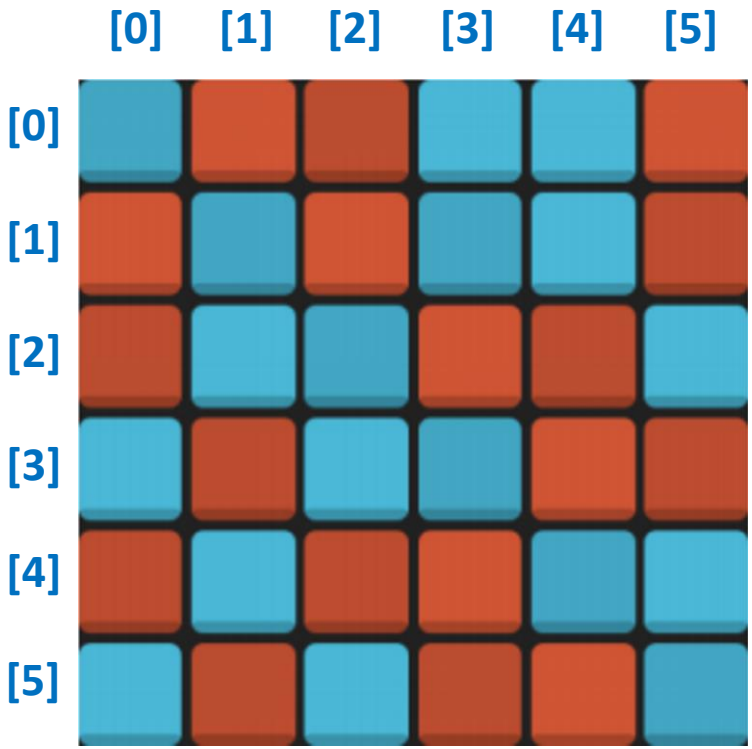
#rows

	[0]	[1]	[2]
[0]	-15	12	13
[1]	12	21	4
[2]	2	-4	3
[3]	-15	23	11

Two Dimensional Arrays

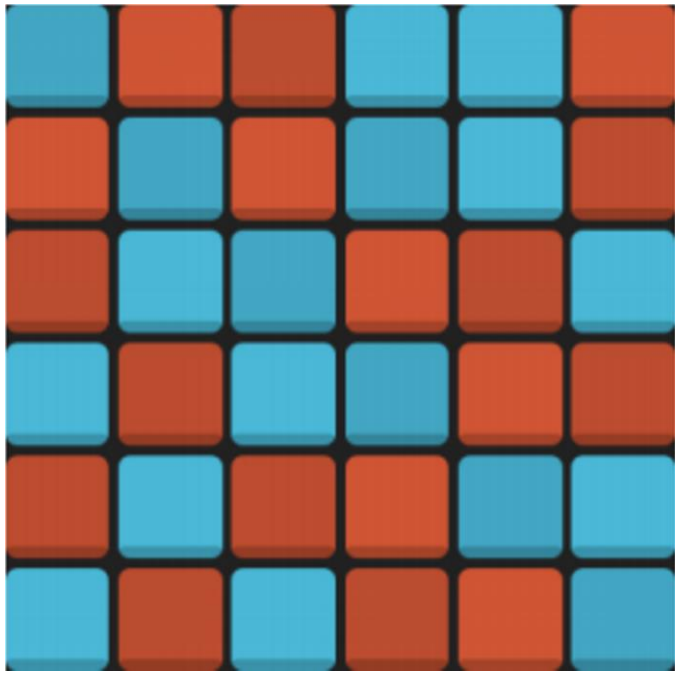
- Used to implement table data

- Example uses:
  - Matrix
  - Image
  - Spreadsheet
  - Game board




```
int board[6][6];
```

0h h1: 2D Array of `int`



```
int board[6][6];
```

rows

columns

Creating the Array

```
int height = 6;
int width = 7;


int board[height][width];
```

Compile Error!

Why?

Creating the Array

```
const int HEIGHT = 6;
const int WIDTH = 7;
```

```
int board[HEIGHT][WIDTH];
```

Initializing the Array

```
const int HEIGHT = 6;
const int WIDTH = 7;

// like 1D arrays, the rest filled with 0
int board[HEIGHT][WIDTH] = { };
```

Initializing the Array

```
const int HEIGHT = 6;
const int WIDTH = 7;

// or we can initialize directly
int board[HEIGHT][WIDTH] = {
                            {1, 2, 3, 4},
                            {1, 4, 9, 16},
                            {1, 8, 27, 64},
                            {1, 16, 81, 256}
                           };
```

Initialize multi-dimensional arrays by grouping together initializers for each dimension

Initializing the Array

```
const int HEIGHT = 6;
const int WIDTH = 7;
const int UNKNOWN = 0;

// or we can initialize with loops!
int board[HEIGHT][WIDTH];

for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        board[row][col] = UNKNOWN;
    }
}
```

i>Clicker #2

```
int board[4][4] = {
                    {1, 2, 3, 4},
                    {1, 4, 9, 16},
                    {1, 8, 27, 64},
                    {1, 16, 81, 256}
                   };
```

What is the value of `board[3][2]`?

A. 8
B. 9
C. 64
D. 81
E. None of the above.

i>Clicker #2

```
int board[4][4] = {
                    {1, 2, 3, 4},
                    {1, 4, 9, 16},
                    {1, 8, 27, 64},
                    {1, 16, 81, 256}
                   };
```

What is the value of `board[3][2]`?

A. 8
B. 9
C. 64
D. 81
E. None of the above.

Clearing a 0h h1 Board

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	0	0	0	0	0
[1]	0	0	0	0	0	0
[2]	0	??	??	??	??	??
[3]	??	??	??	??	??	??
[4]	??	??	??	??	??	??
[5]	??	??	??	??	??	??

UNKNOWN declared within `utility.h`

2	row
0	col

Execution

```
for (int row = 0; row < HEIGHT; row++) {
    for (int col = 0; col < WIDTH; col++) {
        board[row][col] = UNKNOWN;
    }
}
```

i>Clicker #3

```
int main() {
    int array[3][4] = { { 0, 1, 2 },
                        { 3, 4 },
                        { 5 } };
    for (int j = 0; j < 4; j++) {
        int sum = 0;
        for (int i = 0; i < 3; i++) {
            sum += array[i][j];
        }
        cout << sum;
    }
}
```

What prints?

A. 000
B. 245
C. 375
D. 852
E. None of the above

i>Clicker #3

```
int main() {
    int array[3][4] = { { 0, 1, 2 },
                        { 3, 4 },
                        { 5 } };
    for (int j = 0; j < 4; j++) {
        int sum = 0;
        for (int i = 0; i < 3; i++) {
            sum += array[i][j];
        }
        cout << sum;
    }
}
```

sums columns

there are 4 cols

What prints?

A. 000
B. 245
C. 375
D. 852
E. None of the above

i>Clicker #4

Which of the following function declarations are **invalid**?

A. `void printArray(int arr[5][5], int size);`
B. `void printArray(int arr[][5], int size);`
C. `int[5][5] printArray(int arr[5][5], int size);`
D. B and C
E. All of the above

i>Clicker #4

Which of the following function declarations are **invalid**?

Cannot return an array

All but the first dimension is required

A. `void printArray(int arr[5][5], int size);`
B. `void printArray(int arr[][5], int size);`
C. `int[5][5] printArray(int arr[5][5], int size);`
D. B and C
E. All of the above

Today

File I/O

File I/O

(I/O stands for Input/Output)

Handled in C++ via **streams**


DIST. BY CREATORS SYNDICATE
SPEEDBUMP.COM
©2011 COURBELY
7-5
OK, NOW YOU'RE NOT SWIPING AT SALMON. NOW YOU'RE JUST SPLASHING ME.

"AND, AS YOU CAN SEE, WE'VE GOT A STREAM OF HOT AIR COMING FROM THIS DIRECTION."

Stream Characteristics

- Streams flow in one direction

Stream Characteristics

- Program can pull items out of input streams

Stream Characteristics

- Program can pull items out of input streams

```
cin >> x;
```

- Extraction operator

```
>>
```

Stream Characteristics

- Program can put items into output streams

Stream Characteristics

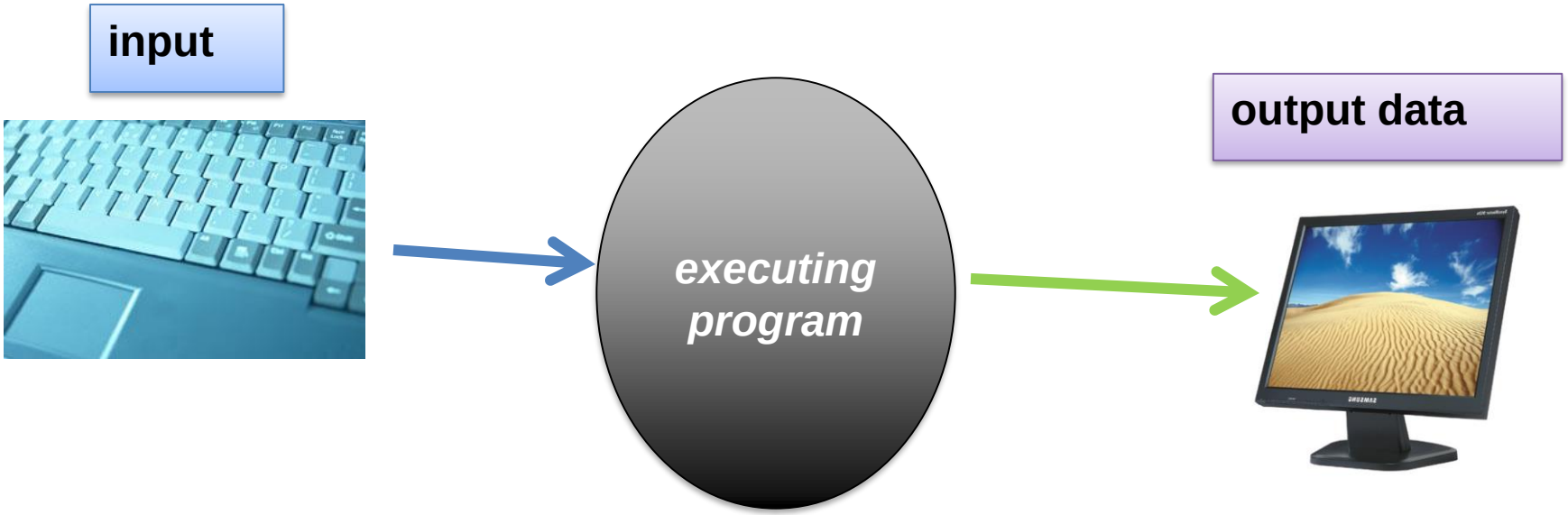
- Program can put items into output streams

```
cout << "output: " << x << endl;
```

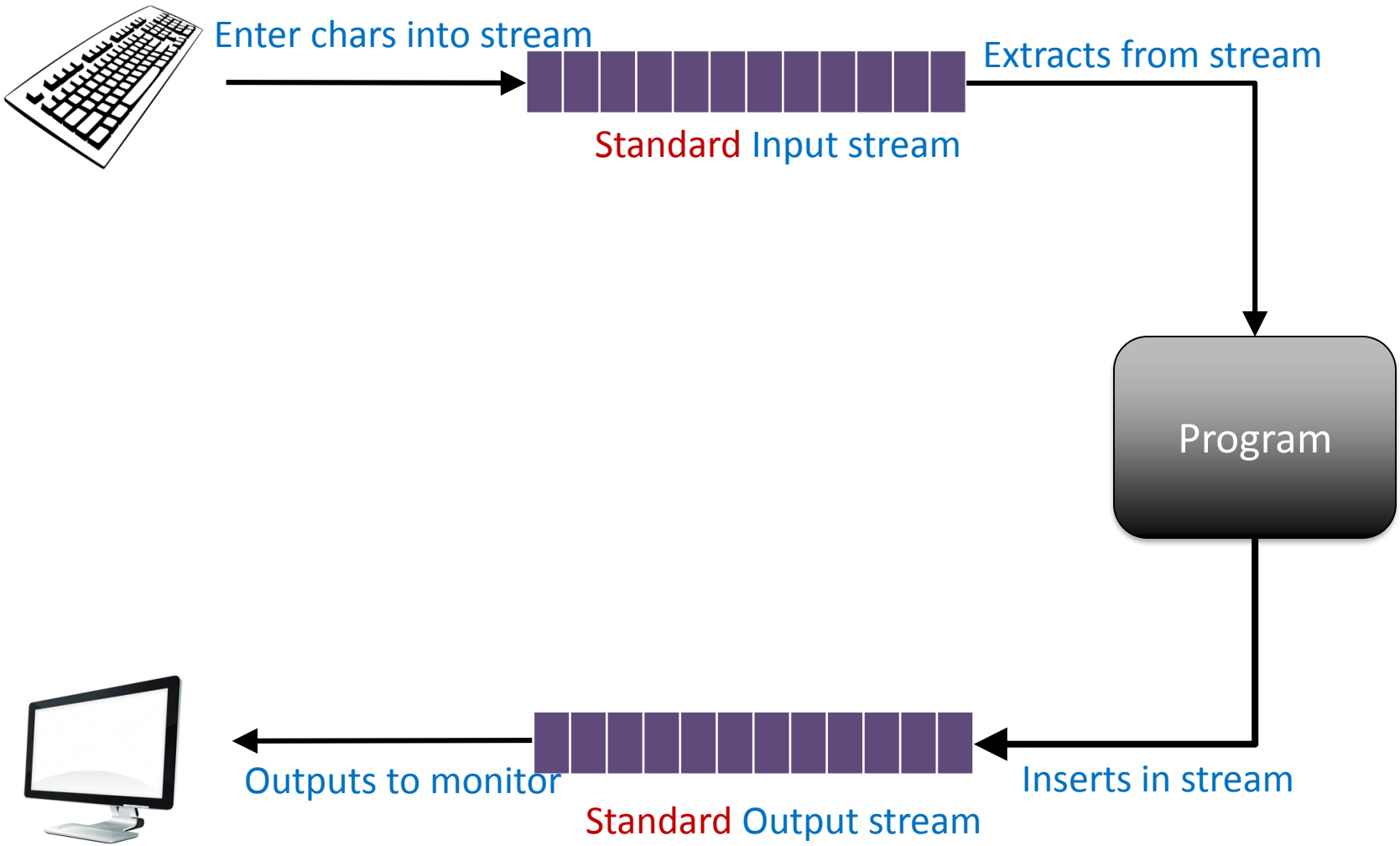
- Insertion operator

```
<<
```

Default Input/Output Streams



Default Input/Output Streams



Fail State

Input: **3 4 5 a 3**

```
int sum = 0;
int count = 0;
int number;

while (cin >> number && number != 0) {
    sum += number;
    count++;
}
cout << static_cast<double>(sum) / count;
if (cin.fail()) {
    cin.clear();
    string str;
    getline(cin, str);
}
```

What does this actually mean?

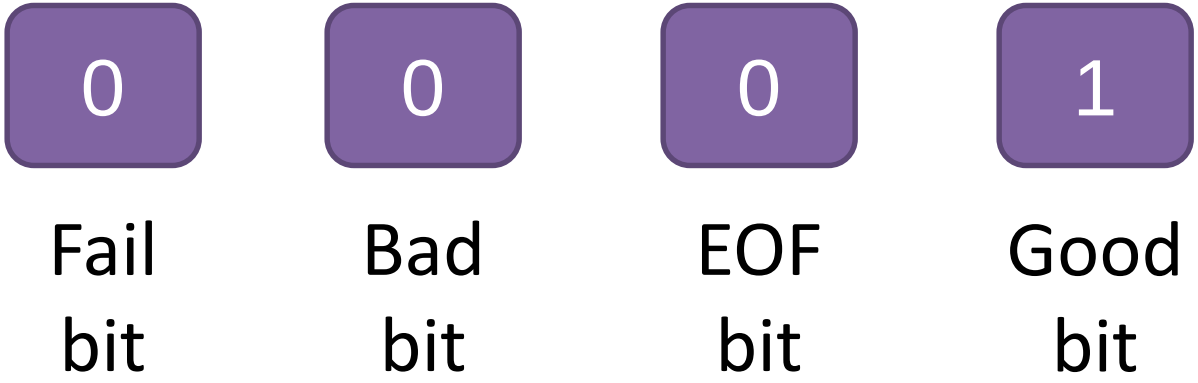
Stream States

- Good  Everything is great!
- Fail  Non-fatal Error - failed to read expected data

  Examples: *failed to convert type*
  or *file does not exist*
- Bad 
- EOF 

State bits

- How does stream object track the stream state?
  - Stores a bit for each state *(i.e., a bool)*
  - When enters a state, bit is set to 1



Example

```
int x = 1;
int y = 2;
int z = 3;
cin >> x;
cin >> y;
cin >> z;
cout << x << ' ' << y << ' ' << z;
```

5	\n	a	\n	7	\n	

Standard Input stream

0	0	0	1
Fail bit	Bad bit	EOF bit	Good bit

Console

Example

Execution

```
int x = 1;
int y = 2;
int z = 3;
cin >> x;
cin >> y;
cin >> z;
cout << x << ' ' << y << ' ' << z;
```

5	\n	a	\n	7	\n	

Standard Input stream

1	x

0	0	0	1
Fail bit	Bad bit	EOF bit	Good bit

Console

Example

Execution

```
int x = 1;
int y = 2;
int z = 3;
cin >> x;
cin >> y;
cin >> z;
cout << x << ' ' << y << ' ' << z;
```

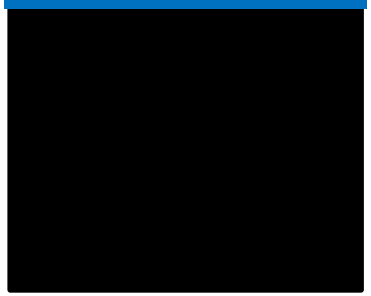
5	\n	a	\n	7	\n	

Standard Input stream

1	x
2	y

0	0	0	1
Fail bit	Bad bit	EOF bit	Good bit

Console



Example

Execution

```
int x = 1;
int y = 2;
int z = 3;
cin >> x;
cin >> y;
cin >> z;
cout << x << ' ' << y << ' ' << z;
```

5	\n	a	\n	7	\n	

Standard Input stream

1	x
2	y
3	z

0	0	0	1
Fail bit	Bad bit	EOF bit	Good bit

Console

Example

```
int x = 1;
int y = 2;
int z = 3;
cin >> x;
cin >> y;
cin >> z;
cout << x << ' ' << y << ' ' << z;
```

Execution

5	\n	a	\n	7	\n	

Standard Input stream

1	x
2	y
3	z

0	0	0	1
Fail bit	Bad bit	EOF bit	Good bit

Console

Example

5 is extracted from input stream

5	\n	a	\n	7	\n	

Standard Input stream

Execution

```
int x = 1;
int y = 2;
int z = 3;
cin >> x;
cin >> y;
cin >> z;
cout << x << ' ' << y << ' ' << z;
```

1	x
2	y
3	z

0	0	0	1
Fail bit	Bad bit	EOF bit	Good bit

Console

Example

5 is extracted from input stream

\n	a	\n	7	\n		

Standard Input stream

Execution

```
int x = 1;
int y = 2;
int z = 3;
cin >> x;
cin >> y;
cin >> z;
cout << x << ' ' << y << ' ' << z;
```

1	x
2	y
3	z

0	0	0	1
Fail bit	Bad bit	EOF bit	Good bit

Console

Example

```
int x = 1;
int y = 2;
int z = 3;
cin >> x;
cin >> y;
cin >> z;
cout << x << ' ' << y << ' ' << z;
```

Execution

\n	a	\n	7	\n		

Standard Input stream

5	x
2	y
3	z

0	0	0	1
Fail bit	Bad bit	EOF bit	Good bit

Console

Example

```
int x = 1;
int y = 2;
int z = 3;
cin >> x;
cin >> y;
cin >> z;
cout << x << ' ' << y << ' ' << z;
```

Execution

\n	a	\n	7	\n		

Standard Input stream

5	x
2	y
3	z

0	0	0	1
Fail bit	Bad bit	EOF bit	Good bit

Console

Example

The leading whitespace (in this case, a newline) is ignored

```
int x = 1;
int y = 2;
int z = 3;
cin >> x;
cin >> y;
cin >> z;
cout << x << ' ' << y << ' ' << z;
```

Execution

\n	a	\n	7	\n		

Standard Input stream

5	x
2	y
3	z

0	0	0	1
Fail bit	Bad bit	EOF bit	Good bit

Console

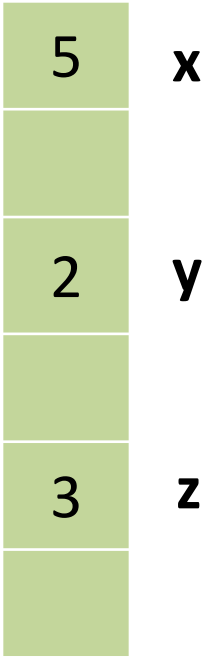
Example

The leading whitespace (in this case, a newline) is ignored

```
int x = 1;
int y = 2;
int z = 3;
cin >> x;
cin >> y;
cin >> z;
cout << x << ' ' << y << ' ' << z;
```

Execution

Standard Input stream



0	0	0	1
Fail bit	Bad bit	EOF bit	Good bit

Console

Example

Whoops, this is not an `int`!

```
int x = 1;
int y = 2;
int z = 3;
cin >> x;
cin >> y;
cin >> z;
cout << x << ' ' << y << ' ' << z;
```

Execution

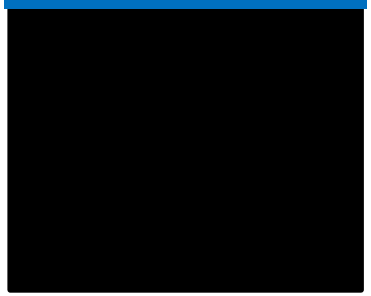
a	\n	7	\n			

Standard Input stream

5	x
2	y
3	z

0	0	0	1
Fail bit	Bad bit	EOF bit	Good bit

Console



Example

Whoops, this is not an `int`!

a	\n	7	\n			

```
int x = 1;
int y = 2;
int z = 3;
cin >> x;
cin >> y;
cin >> z;
cout << x << ' ' << y << ' ' << z;
```

Execution

Stream enters **fail** state.

NO reading takes place

5	x
2	y
3	z

1	0	0	0
Fail bit	Bad bit	EOF bit	Good bit

Console

Example

Whoops, this is not an `int`!

a	\n	7	\n			

```
int x = 1;
int y = 2;
int z = 3;
cin >> x;
cin >> y;
cin >> z;
cout << x << ' ' << y << ' ' << z;
```

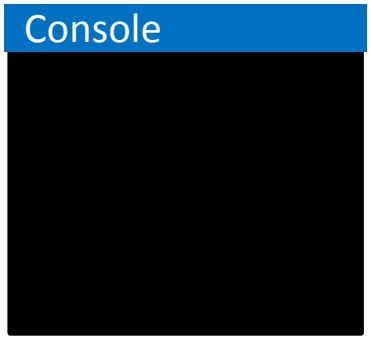
Execution

C++ 11
will replace current value
of y with a 0

5	x
0	y
3	z

1	0	0	0
Fail bit	Bad bit	EOF bit	Good bit

Console



Example

Whoops, this is not an `int`!

a	\n	7	\n			

```
int x = 1;
int y = 2;
int z = 3;
cin >> x;
cin >> y;
cin >> z;
cout << x << ' ' << y << ' ' << z;
```

Execution

Older compilers will leave current value of y

5	x
2	y
3	z

1	0	0	0
Fail bit	Bad bit	EOF bit	Good bit

Console

Example

```
int x = 1;
int y = 2;
int z = 3;
cin >> x;
cin >> y;
cin >> z;
cout << x << ' ' << y << ' ' << z;
```

Execution

a	\n	7	\n			

Standard Input stream

5	x
0	y
3	z

Fail bit stays true until you call **`cin.clear()`**!

1	0	0	0
Fail bit	Bad bit	EOF bit	Good bit

Console

Example

Nothing changes due to fail state

a	\n	7	\n			

```
int x = 1;
int y = 2;
int z = 3;
cin >> x;
cin >> y;
cin >> z;
cout << x << ' ' << y << ' ' << z;
```

Execution

Stream still in fail state.

No reading takes place, even with C++11

5	x
0	y
3	z

1	0	0	0
Fail bit	Bad bit	EOF bit	Good bit

Console

Example

Nothing changes due to fail state

a	\n	7	\n			

```
int x = 1;
int y = 2;
int z = 3;
cin >> x;
cin >> y;
cin >> z;
cout << x << ' ' << y << ' ' << z;
```

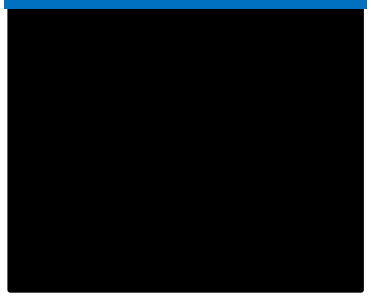
Execution

Since
1) Stream still in fail state
2) no reading takes place
the value of 'z' is not altered

5	x
0	y
3	z

1	0	0	0
Fail bit	Bad bit	EOF bit	Good bit

Console



Example

```
int x = 1;
int y = 2;
int z = 3;
cin >> x;
cin >> y;
cin >> z;
cout << x << ' ' << y << ' ' << z;
```

Execution

a	\n	7	\n			

Standard Input stream

5	x
0	y
3	z

1	0	0	0
Fail bit	Bad bit	EOF bit	Good bit

Console

```
5 0 3
```

Example

```
int x = 1;
int y = 2;
int z = 3;
cin >> x;
cin >> y;
cin >> z;
cout << x << ' ' << y << ' ' << z;
```

Execution

a	\n	7	\n			

Output for Visual Studio

5	x
2	y
3	z

1	0	0	0
Fail bit	Bad bit	EOF bit	Good bit

Console

```
5 2 3
```

Example

```
int x = 1;
int y = 2;
int z = 3;
cin >> x;
cin >> y;
cin >> z;
cout << x << ' ' << y << ' ' << z;
```

Execution

a	\n	7	\n			

Output for Xcode
and g++ 4.8.0 (C++ 11)
and autograder

5	x
0	y
3	z

1	0	0	0
Fail bit	Bad bit	EOF bit	Good bit

Console
5 0 3

i>Clicker #5: What prints? Assume C++11

```
int x = 1;
int y = 2;
char ch = '#';
cin >> x;
cin >> y;
cin >> ch;
cout << x << ' ' << y << ' ' << ch;
```

5	\n	a	\n	7	\n	

Standard Input stream

1	x
2	y
'#'	ch

0	0	0	1
Fail bit	Bad bit	EOF bit	Good bit

A. 5 0 #
B. 5 0 a
C. 5 0 \n
D. None of above

i>Clicker #5: What prints? Assume C++11

```
int x = 1;
int y = 2;
char ch = '#';
cin >> x;
cin >> y;
cin >> ch;
cout << x << ' ' << y << ' ' << ch;
```

5	\n	a	\n	7	\n	

Standard Input stream

cin goes into 'fail' state –
no further reading takes place

5	x
0	y
'#'	ch

1	0	0	0
Fail bit	Bad bit	EOF bit	Good bit

A. 5 0 #
B. 5 0 a
C. 5 0 \n
D. None of above

Moral of the Story

- **DO NOT** trust input from a stream in a **fail** state.
  - That stream no longer works for any further input
  - The program **DOES NOT** stop or give an error message
  - The variables become undefined; their values depend on the compiler

Checking the Fail Bit

- To check fail bit, use **`cin.fail()`**
- **For example, to <u>read until</u> the first <u>non-integer:</u>**

```
int x;
cin >> x;
while (!cin.fail()) {
    // do stuff
    cin >> x;
}
```

i>Clicker #6

- What happens in this code?

```
// user input: 3.14 pies
int n;
string s;
cin >> n >> s;
cin >> s >> n;
cout << n << s;
```

A. "3pies" is printed
B. "14pies" is printed
C. cin goes into a fail state
D. The program pauses
E. Both A and C

i>Clicker #6

- What happens in this code?

```
// user input: 3.14 pies
int n;
string s;
cin >> n >> s;
cin >> s >> n;
cout << n << s;
```

A. "3pies" is printed
B. "14pies" is printed
C. cin goes into a fail state
D. The program pauses
E. Both A and C

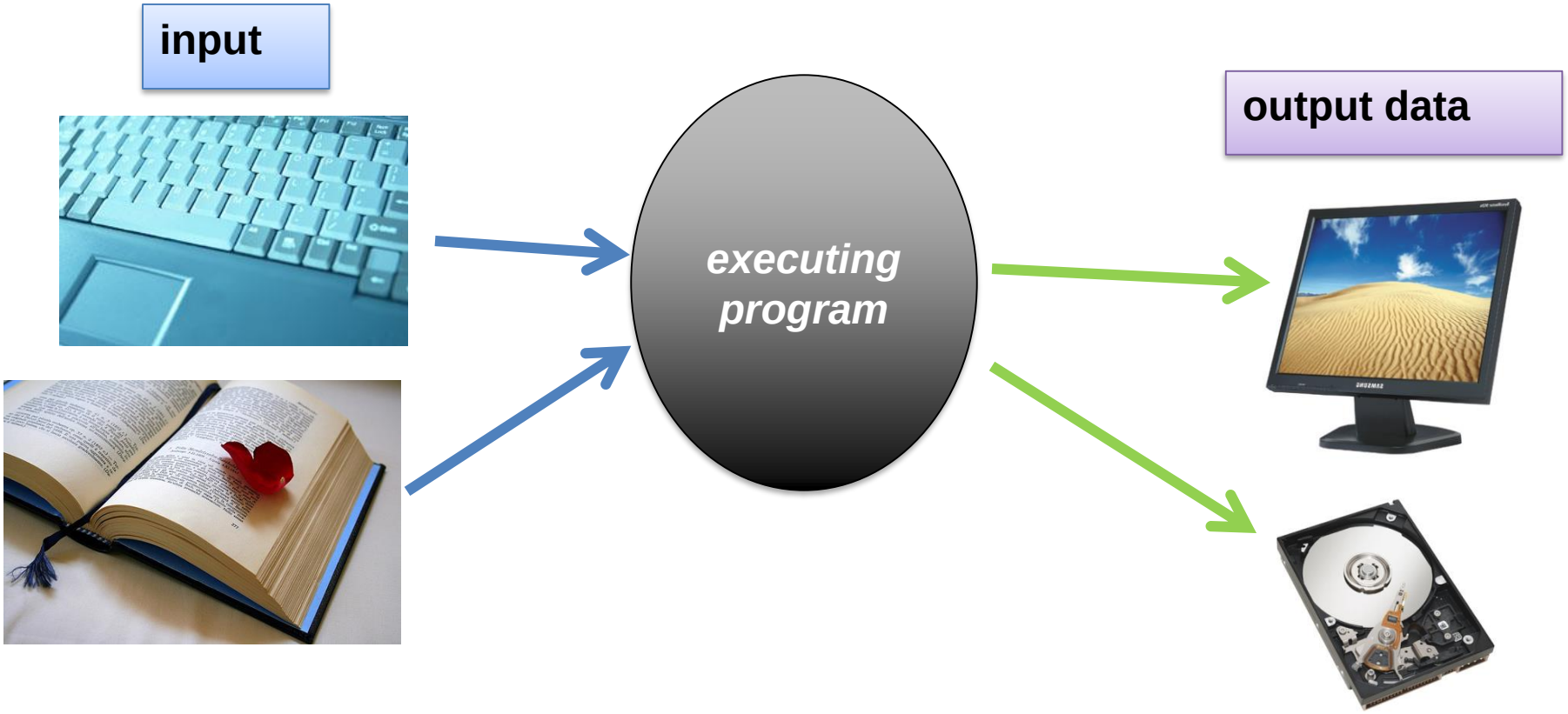
Intermission

The P3 autograder doesn’t directly read in boards, but reads in a file with filenames, then reads those files for the boards.

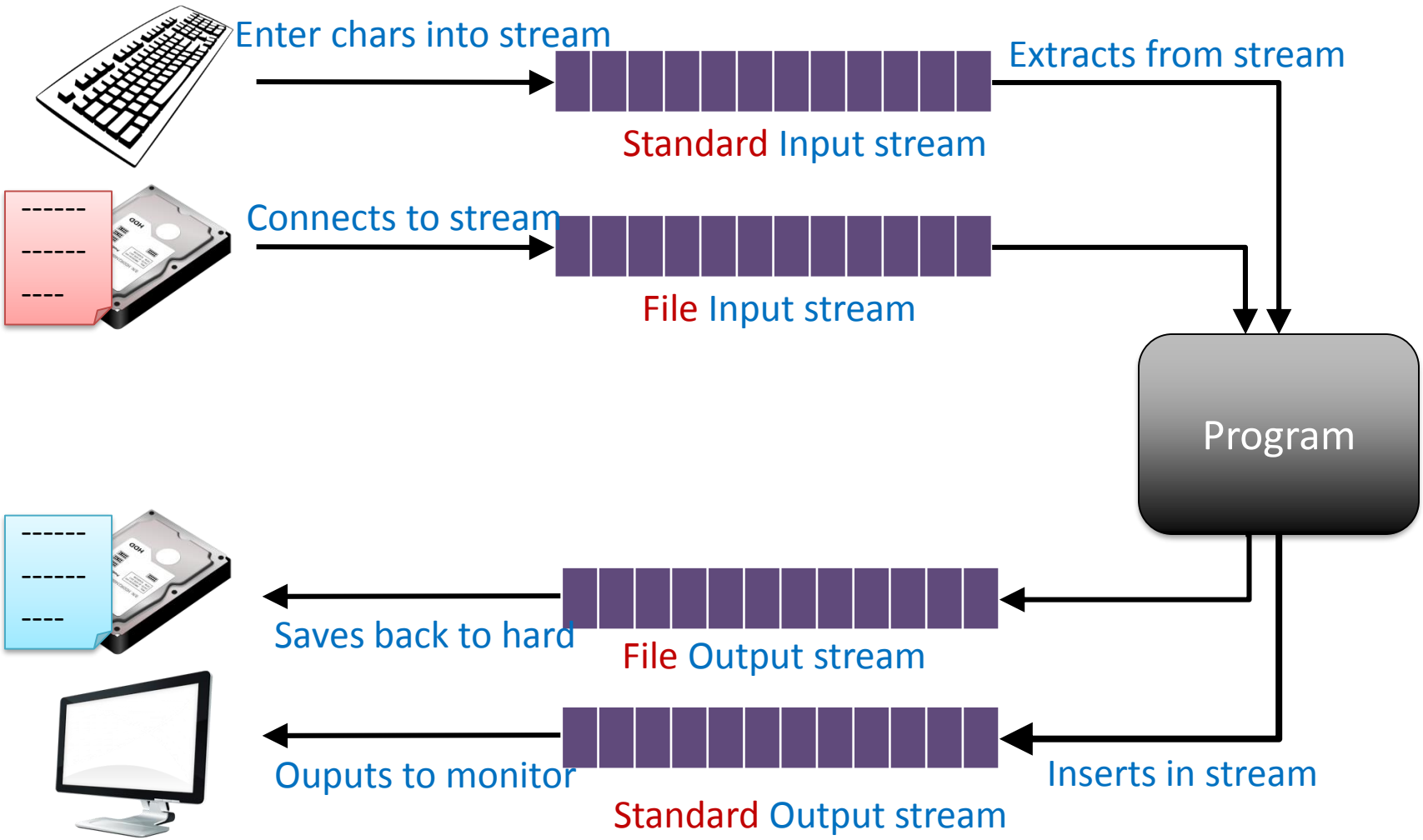
Digital Divide

Duet

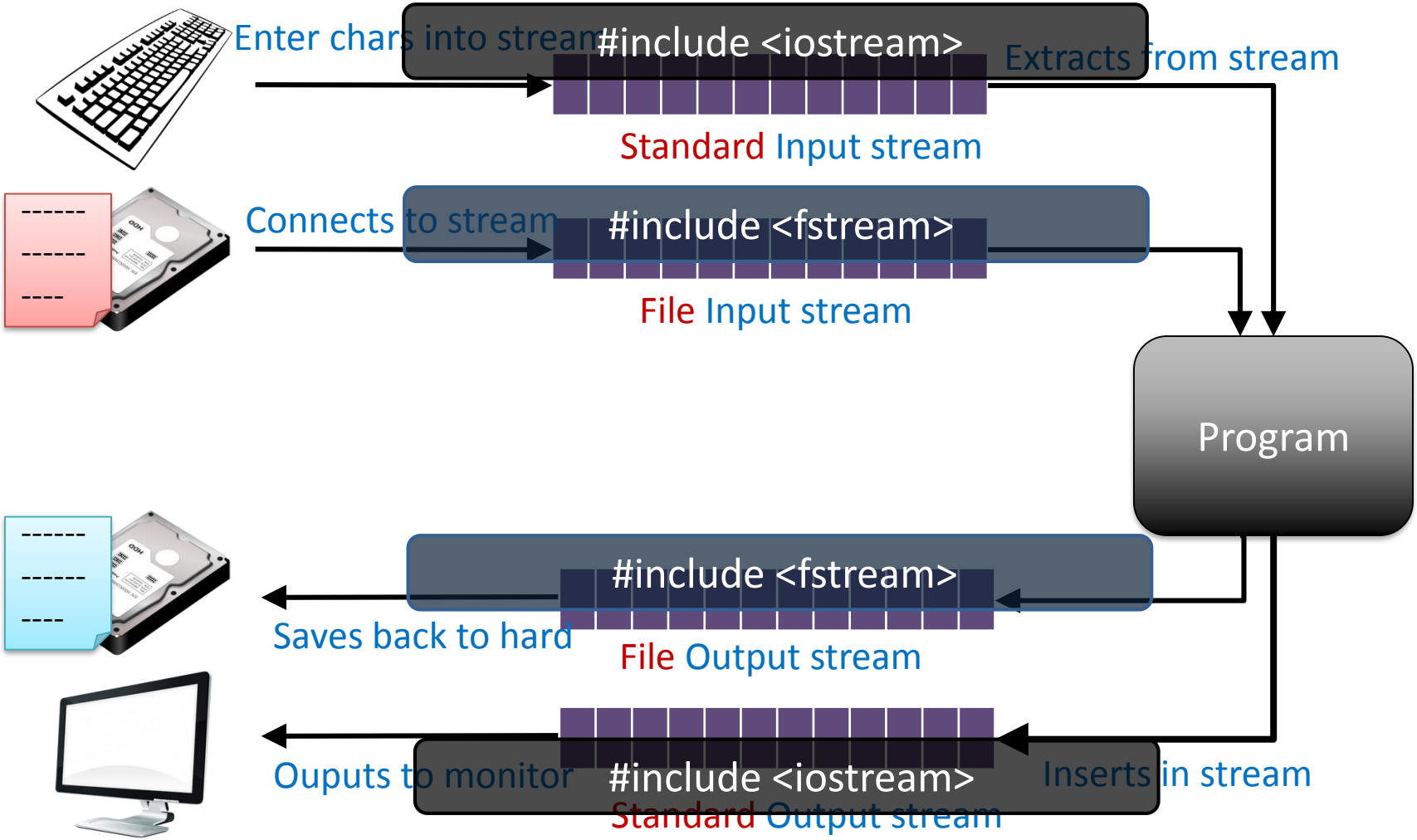
Multiple Input/Output Streams



Multiple Input/Output Streams



Multiple Input/Output Streams



<iostream> vs <fstream>

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

int main() {
    int x;



    cin >> x;

}
```

<iostream> vs <fstream>

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

int main() {
    int x;


    cin >> x;

}
```

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

int main() {
    int x;
    ifstream input_file;
    input_file.open("filename");
    input_file >> x;
    input_file.close();
}
```

<iostream> vs <fstream>

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

int main() {
    int x;
    ifstream input_file;
    input_file.open("filename");
    input_file >> x;
    input_file.close();
}
```

Reading from Files

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

int main() {
    int x;
    ifstream input_file;
    input_file.open("filename");
    input_file >> x;
    input_file.close();
}
```

Reading from Files

```
#include <fstream>
```

- The **`fstream`** library contains the datatype **`ifstream`** (for reading from files)

Reading from Files

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

int main() {
    int x;
    ifstream input_file;
    input_file.open("filename");
    input_file >> x;
    input_file.close();
}
```

Reading from Files

```
ifstream input_file;
```

datatype

variable name

- Declares an input stream variable
  - `ifstream` is a datatype, just like `int`, `bool`, etc.

Reading from Files

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

int main() {
    int x;
    ifstream input_file;
    input_file.open("filename");
    input_file >> x;
    input_file.close();
}
```

Reading from Files

```
input_file.open("filename");
```

variable name

Identifier in program

File name

What it's called on HD

- Tells C++ which file to open, and prepares the file to be read

Reading from Files

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

int main() {
    int x;
    ifstream input_file;
    input_file.open("filename");
    input_file >> x;
    input_file.close();
}
```

Reading from Files

```
input_file >> x;
```

- Just like with `cin`, this reads a single integer from the file

Reading from Files

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

int main() {
    int x;
    ifstream input_file;
    input_file.open("filename");
    input_file >> x;
    input_file.close();
}
```

Reading from Files

```
input_file.close();
```

- Tells C++ to close the file, so it cannot be read from again

What about writing to files?

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

int main() {
    int x = 42;



    cout << x;

}
```

What about writing to files?

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

int main() {
    int x = 42;
    ofstream output_file;
    output_file.open("filename");
    output_file << x;
    output_file.close();
}
```

<iostream> vs <fstream>

- #include<iostream>

- #include<fstream>

<iostream> vs <fstream>

- #include<iostream>
- cin is already defined

- #include<fstream>
- Must declare variable first of type ifstream
  ifstream in_file;

<iostream> vs <fstream>

- #include<iostream>
- cin is already defined
- cin only works on keyboard input

- #include<fstream>
- Must declare variable first of type ifstream
- Must use open() to specify file to read/write

```
in_file.open("board.txt");
```

<iostream> vs <fstream>

- `#include<iostream>`
- `cin` is already defined
- `cin` only works on keyboard input
- read into variable
  `cin >> x;`

- `#include<fstream>`
- Must declare variable first of type `ifstream`
- Must use `open()` to specify file to read/write
- read into variable
  `in_file >> x;`

<iostream> vs <fstream>

- `#include<iostream>`
- `cin` is already defined
- `cin` only works on keyboard input
- read into variable
- No extra step necessary

- `#include<fstream>`
- Must declare variable first of type `ifstream`
- Must use `open()` to specify file to read/write
- read into variable
- Must use `close()` to stop reading/writing file. `in_file.close();`

What about the other bits?

- Good 

  Everything is great!

- Fail 

  Non-fatal Error - failed to read expected data

  Examples: *failed to convert type*
  or *file does not exist*

- Bad 

  Fatal Error - stream can no longer be used

  Example: *unplugging a USB drive*
  loose network connect

- EOF 

  End of the stream encountered

EOF Example

- Set when the entire file has been read.

- Unsetting **depends on the system**
  - Could be unset automatically on the next read (even if the read was unsuccessful!)

  - Could require it to be unset manually with `clear()`

EOF Example

```
int i = 0, j = 0;
char ch = ' ', k = ' ';
ifstream inFile;
inFile.open("input_file");
inFile >> i;
inFile >> j;
inFile >> ch;
inFile >> k;
```


0	0	0	1
Fail bit	Bad bit	EOF bit	Good bit

EOF Example

Execution

```
int i = 0, j = 0;
char ch = ' ', k = ' ';
ifstream inFile;
inFile.open("input_file");
inFile >> i;
inFile >> j;
inFile >> ch;
inFile >> k;
```

0	i
0	j

0	0	0	1
Fail bit	Bad bit	EOF bit	Good bit

EOF Example

Execution

```
int i = 0, j = 0;
char ch = ' ', k = ' ';
ifstream inFile;
inFile.open("input_file");
inFile >> i;
inFile >> j;
inFile >> ch;
inFile >> k;
```

0	i
0	j
''	ch
''	k

0	0	0	1
Fail bit	Bad bit	EOF bit	Good bit

EOF Example

Execution

```
int i = 0, j = 0;
char ch = ' ', k = ' ';
ifstream inFile;
inFile.open("input_file");
inFile >> i;
inFile >> j;
inFile >> ch;
inFile >> k;
```

0	i
0	j
''	ch
''	k
???	inFile

0	0	0	1
Fail bit	Bad bit	EOF bit	Good bit

EOF Example

```
int i = 0, j = 0;
char ch = ' ', k = ' ';
ifstream inFile;
inFile.open("input_file");
inFile >> i;
inFile >> j;
inFile >> ch;
inFile >> k;
```

Execution

0	i
0	j
''	ch
''	k
	inFile

15
5
Q
<eof>

0	0	0	1
Fail bit	Bad bit	EOF bit	Good bit

EOF Example

```
int i = 0, j = 0;
char ch = ' ', k = ' ';
ifstream inFile;
inFile.open("input_file");
inFile >> i;
inFile >> j;
inFile >> ch;
inFile >> k;
```

Execution

0	i
0	j
''	ch
''	k
	inFile

15
5
Q
<eof>

0	0	0	1
Fail bit	Bad bit	EOF bit	Good bit

EOF Example

```
int i = 0, j = 0;
char ch = ' ', k = ' ';
ifstream inFile;
inFile.open("input_file");
inFile >> i;
inFile >> j;
inFile >> ch;
inFile >> k;
```

Execution

0	i
0	j
''	ch
''	k
	inFile

15
5
Q
<eof>

0	0	0	1
Fail bit	Bad bit	EOF bit	Good bit

EOF Example

```
int i = 0, j = 0;
char ch = ' ', k = ' ';
ifstream inFile;
inFile.open("input_file");
inFile >> i;
inFile >> j;
inFile >> ch;
inFile >> k;
```

Execution

15	i
0	j
''	ch
''	k
	inFile

15
5
Q
<eof>

0	0	0	1
Fail bit	Bad bit	EOF bit	Good bit

EOF Example

```
int i = 0, j = 0;
char ch = ' ', k = ' ';
ifstream inFile;
inFile.open("input_file");
inFile >> i;
inFile >> j;
inFile >> ch;
inFile >> k;
```

Execution

15	i
5	j
''	ch
''	k
	inFile

15
5
Q
<eof>

0	0	0	1
Fail bit	Bad bit	EOF bit	Good bit

EOF Example

```
int i = 0, j = 0;
char ch = ' ', k = ' ';
ifstream inFile;
inFile.open("input_file");
inFile >> i;
inFile >> j;
inFile >> ch;
inFile >> k;
```

Execution

15	i
5	j
'Q'	ch
' '	k
	inFile

15
5
Q
<eof>

0	0	0	1
Fail bit	Bad bit	EOF bit	Good bit

EOF Example

```
int i = 0, j = 0;
char ch = ' ', k = ' ';
ifstream inFile;
inFile.open("input_file");
inFile >> i;
inFile >> j;
inFile >> ch;
inFile >> k;
```

Execution

15	i
5	j
'Q'	ch
' '	k
	inFile

15
5
Q
<eof>

0	0	0	1
Fail bit	Bad bit	EOF bit	Good bit

EOF Example

```
int i = 0, j = 0;
char ch = ' ', k = ' ';
ifstream inFile;
inFile.open("input_file");
inFile >> i;
inFile >> j;
inFile >> ch;
inFile >> k;
```

Execution

15	i
5	j
'Q'	ch
' '	k
	inFile

15
5
Q
<eof>

0	0	0	1
Fail bit	Bad bit	EOF bit	Good bit

EOF Example

```
int i = 0, j = 0;
char ch = ' ', k = ' ';
ifstream inFile;
inFile.open("input_file");
inFile >> i;
inFile >> j;
inFile >> ch;
inFile >> k;
```

Execution

15	i
5	j
'Q'	ch
' '	k
	inFile

15
5
Q
<eof>

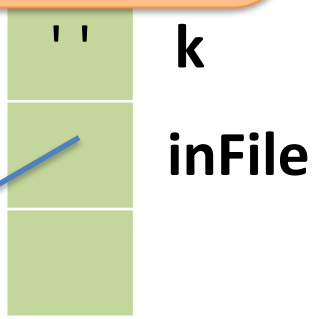
1	0	1	0
Fail bit	Bad bit	EOF bit	Good bit

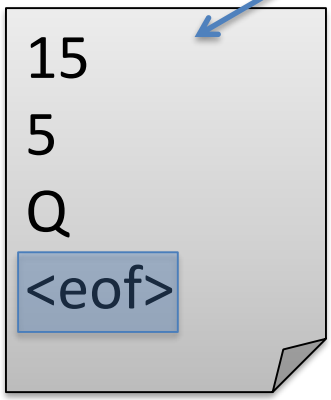
EOF Example

```
int i = 0, j = 0;
char ch = ' ', k = ' ';
ifstream inFile;
inFile.open("input_file");
inFile >> i;
inFile >> j;
inFile >> ch;
inFile >> k;
```

Execution

But watch out!
In some compilers, if you try to read in again…





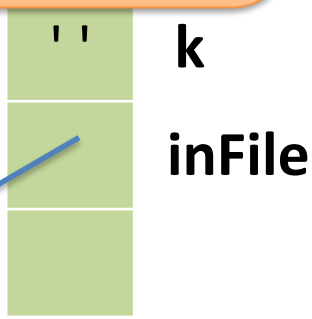
1	0	1	0
Fail bit	Bad bit	EOF bit	Good bit

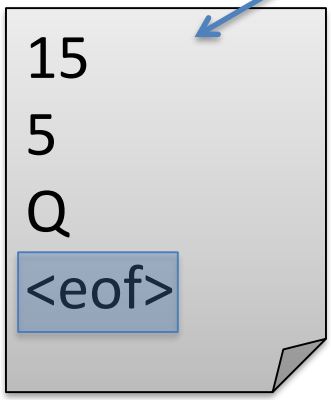
EOF Example

```
int i = 0, j = 0;
char ch = ' ', k = ' ';
ifstream inFile;
inFile.open("input_file");
inFile >> i;
inFile >> j;
inFile >> ch;
inFile >> k;
inFile >> i;
```

Execution

But watch out!
In some compilers, if you try to read in again…





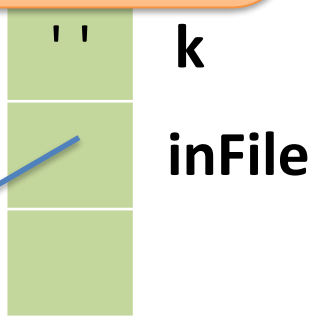
1	0	1	0
Fail bit	Bad bit	EOF bit	Good bit

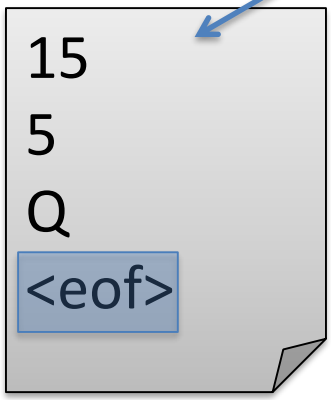
EOF Example

```
int i = 0, j = 0;
char ch = ' ', k = ' ';
ifstream inFile;
inFile.open("input_file");
inFile >> i;
inFile >> j;
inFile >> ch;
inFile >> k;
inFile >> i;
```

Execution

But watch out!
In some compilers, if you try to read in again…





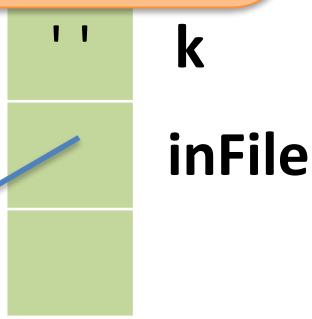
1	0	1	0
Fail bit	Bad bit	EOF bit	Good bit

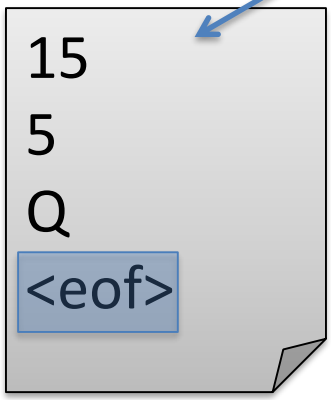
EOF Example

```
int i = 0, j = 0;
char ch = ' ', k = ' ';
ifstream inFile;
inFile.open("input_file");
inFile >> i;
inFile >> j;
inFile >> ch;
inFile >> k;
inFile >> i;
```

Execution

But watch out!
In some compilers, if you try to read in again…





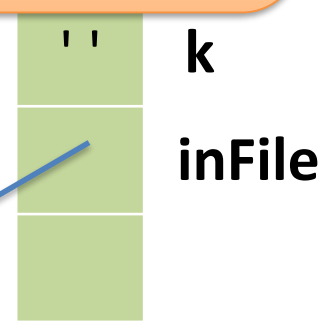
1	0	0	0
Fail bit	Bad bit	EOF bit	Good bit

EOF Example

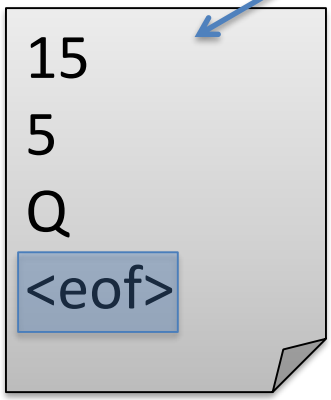
```
int i = 0, j = 0;
char ch = ' ', k = ' ';
ifstream inFile;
inFile.open("input_file");
inFile >> i;
inFile >> j;
inFile >> ch;
inFile >> k;
inFile >> i;
```

Execution

But watch out!
In some compilers, if you try to read in again…
The EOF bit resets!



1	0	0	0
Fail bit	Bad bit	EOF bit	Good bit



Moral of the Story

- **DO NOT** use `cin.eof()` or `ifstream.eof()` as a condition

- Use these instead:

```
while (inFile >> x) { ... }
while (!inFile.fail()) { ... }
```

Resetting Streams

- If `cin` or `ifstream` goes into a fail state, what do we do?
  - Use **`cin.clear()`** or **`ifstream.clear()`** to reset it

- We can wrap it all up in a function

A Function to do the Work

```
void openFile(ifstream &ins) {
    string fileName;
    cout << "Enter filename: ";
    cin >> fileName;
    ins.open(fileName);
    while (!ins.good()) {
        ins.clear();
        cout << "Error in opening file";
        cout << "Enter filename: ";
        cin >> fileName;
        ins.open(fileName);
    }
}
```

A Function to do the Work

note ifstream is passed-by-reference

```
void openFile(ifstream &ins) {
    string fileName;
    cout << "Enter filename: ";
    cin >> fileName;
    ins.open(fileName);
    while (!ins.good()) {
        ins.clear();
        cout << "Error in opening file";
        cout << "Enter filename: ";
        cin >> fileName;
        ins.open(fileName);
    }
}
```

A Function to do the Work

```
void openFile(ifstream &ins) {
    string fileName;
    cout << "Enter filename: ";
    cin >> fileName;
    ins.open(fileName);
    while (!ins.good()) {
        ins.clear();
        cout << "Error in opening file";
        cout << "Enter filename: ";
        cin >> fileName;
        ins.open(fileName);
    }
}
```

get the name

A Function to do the Work

```
void openFile(ifstream &ins) {
    string fileName;
    cout << "Enter filename: ";
    cin >> fileName;
    ins.open(fileName);
    while (!ins.good()) {
        ins.clear();
        cout << "Error in opening file";
        cout << "Enter filename: ";
        cin >> fileName;
        ins.open(fileName);
    }
}
```

open stream using that name

A Function to do the Work

```
void openFile(ifstream &ins) {
    string fileName;
    cout << "Enter filename: ";
    cin >> fileName;
    ins.open(fileName);
    while (!ins.good()) {
        ins.clear();
        cout << "Error in opening file";
        cout << "Enter filename: ";
        cin >> fileName;
        ins.open(fileName);
    }
}
```

check to see if it opened

A Function to do the Work

```
void openFile(ifstream &ins) {
    string fileName;
    cout << "Enter filename: ";
    cin >> fileName;
    ins.open(fileName);
    while (!ins.good()) {
        ins.clear();
        cout << "Error in opening fil
        cout << "Enter filename: ";
        cin >> fileName;
        ins.open(fileName);
    }
}
```

check to see
if it opened
No issue with
"clearing" a stream
that is already in a
"good" state

A Function to do the Work

```
void openFile(ifstream &ins) {
    string fileName;
    cout << "Enter filename: ";
    cin >> fileName;
    ins.open(fileName);
    while (!ins.good()) {
        ins.clear();
        cout << "Error in opening file";
        cout << "Enter filename: ";
        cin >> fileName;
        ins.open(fileName);
    }
}
```

Error message

A Function to do the Work

```
void openFile(ifstream &ins) {
    string fileName;
    cout << "Enter filename: ";
    cin >> fileName;
    ins.open(fileName);
    while (!ins.good()) {
        ins.clear();
        cout << "Error in opening file";
        cout << "Enter filename: ";
        cin >> fileName;
        ins.open(fileName);
    }
}
```

get the name again

A Function to do the Work

```
void openFile(ifstream &ins) {
    string fileName;
    cout << "Enter filename: ";
    cin >> fileName;
    ins.open(fileName);
    while (!ins.good()) {
        ins.clear();
        cout << "Error in opening file";
        cout << "Enter filename: ";
        cin >> fileName;
        ins.open(fileName);
    }
}
```

open stream using that name

A Function to do the Work

```
void openFile(ifstream &ins) {
    string fileName;
    cout << "Enter filename: ";
    cin >> fileName;
    ins.open(fileName);
    while (!ins.good()) {
        ins.clear();
        cout << "Error in opening file";
        cout << "Enter filename: ";
        cin >> fileName;
        ins.open(fileName);
    }
}
```

check again; if still bad, reset

A Function to do the Work

```
void openFile(ifstream &ins) {
    string fileName;
    cout << "Enter filename: ";
    cin >> fileName;
    ins.open(fileName);
    while (!ins.good()) {
        ins.clear();
        cout << "Error in opening file";
        cout << "Enter filename: ";
        cin >> fileName;
        ins.open(fileName);
    }
}
```

only way out is for file to open

Summary

`cin` `cout` `ifstream` `ofstream`

- All of the above have states:
  - The **fail** state means a failure to convert types
  - The **EOF** state means the end of the file has been reached
- Check for these with **`stream_name.fail()`**
  - **DO NOT** use `stream_name.eof()`
- Use **`cin.clear()`** and **`stream_name.clear()`** to reset the states

Next Class: Classes!

(Custom Datatypes)