#JustWebinarThings

- Please set your chat to “All panelists and attendees” instead of “All panelists,” so that you all can see each other’s messages.
- You should all have the ability to unmute now!
- Ask your questions onto the chat on the webinar Q&A, or live!
- We now have a poll for non-verbal feedback, including yes, no, faster, slower, coffee break, etc.

Streams

CS 106L Spring 2020 – Avery Wang and Anna Zeng
Stanford University

15 April 2020

Game Plan

- overview
- input/output streams
- announcements
- stream internals
- (stream manipulators)
- (stringstream)

Recap

So Far…

Introductory C++ miscellany:

- Types: pair, struct, auto
- Uniform initialization
- References

References

```
// non-const reference to each element in courses
for (auto& [code, time, instructors] : courses) {
      code = “42L”;
}

// const reference to each element in courses
for (const auto& [code, time, instructors] : courses) {
      code = “42L”;  // compiler error!
}
```

References

```
// non-const reference to each element in courses
for (auto& [code, time, instructors] : courses) {
     code = “42L”;
}

// const reference to each element in courses
for (const auto& [code, time, instructors] : courses) {
     code = “42L”;  // compiler error!
}

// references in parameters and return values

ostream& operator<<(ostream& os, const string& rhs);
```

References

```
// non-const reference to each element in courses
for (auto& [code, time, instructors] : courses) {
      code = “42L”;
}

// const reference to each element in courses
for (const auto& [code, time, instructors] : courses) {
      code = “42L”;  // compiler error!
}

// references in parameters and return values

ostream& operator<<(ostream& os, const string& rhs);
```

Streams

Streams - Introduction

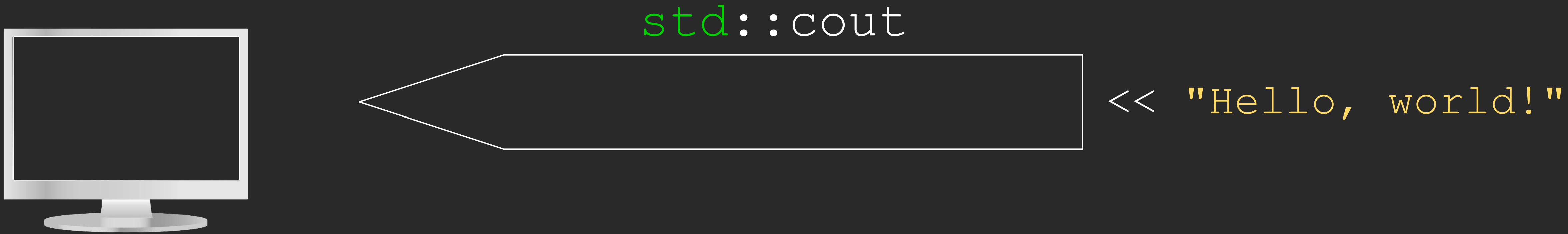
A stream is an abstraction for input/output.

You can think of it as a source (input) or destination (output) of characters of indefinite length.

Streams - Introduction

A stream is an abstraction for input/output.

You can think of it as a source (input) or destination (output) of characters of indefinite length.

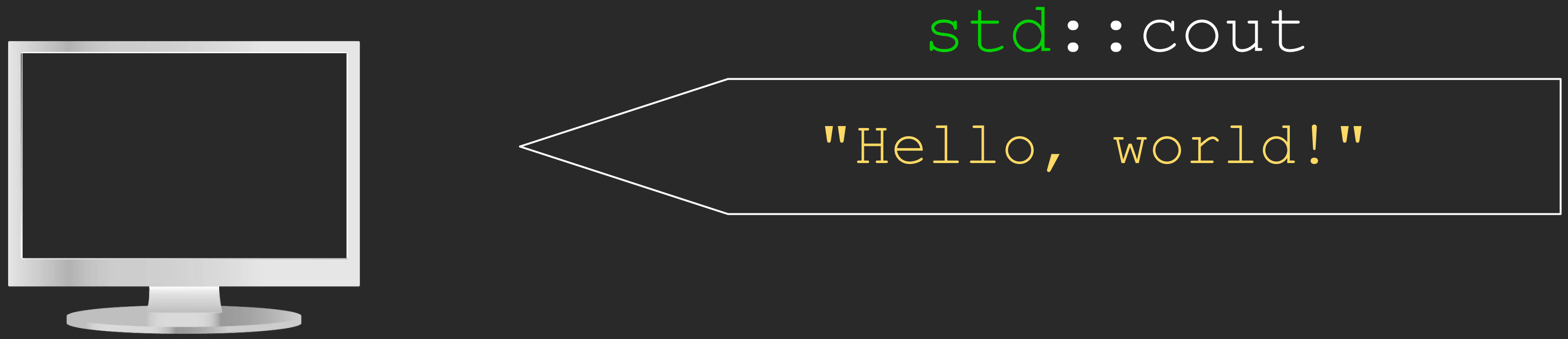


```
std::cout
<< "Hello, world!"
```

Streams - Introduction

A stream is an abstraction for input/output.

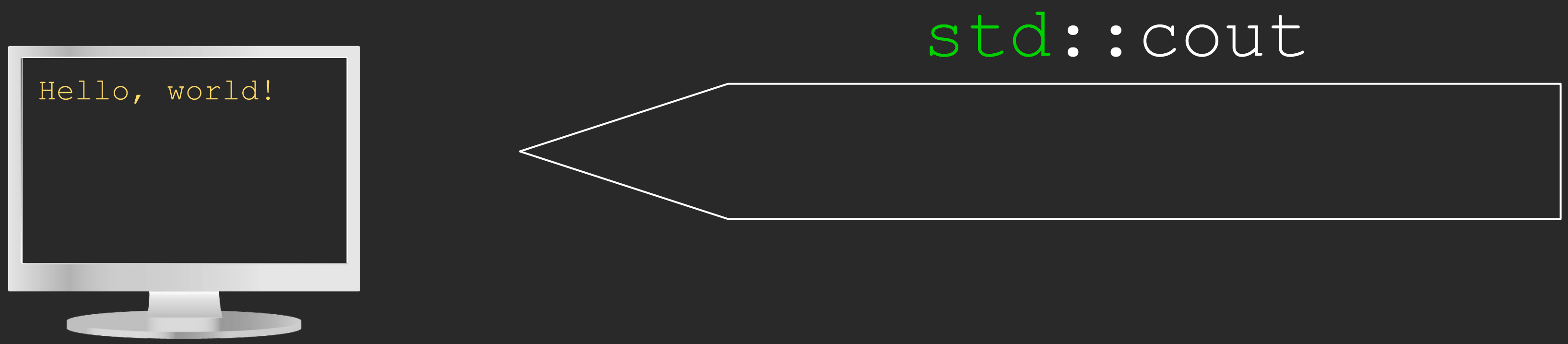
You can think of it as a source (input) or destination (output) of characters of indefinite length.



Streams - Introduction

A stream is an abstraction for input/output.

You can think of it as a source (input) or destination (output) of characters of indefinite length.



The Idea Behind Streams

The Idea Behind Streams

You can write data of multiple types to stream objects.

```
cout << "Strings work!" << endl;
cout << 1729 << endl;
cout << 3.14 << endl;
cout << "Mixed types - " << 1123 << endl;
```

In particular, any primitive type can be inserted into a stream! For other types, you need to explicitly tell C++ how to do this.

The Idea Behind Streams

How does this work?

Idea:

- Input from user is in text form (`string`)
- Output to user is in text form (`string`)
- Intermediate computation needs to be done on object type

The Idea Behind Streams

Streams allow a C++ programmer to convert between the string representation of data, and the data itself.

Types of Streams

Output Streams

Can only receive data.

- The `std::cout` stream is an example of an output stream.
- All output streams are of type `std::ostream`.

Send data using stream insertion operator: <<

Insertions converts data to string and sends to stream.

Output Streams

We can use a `std::ostream` for more than just printing to a console.

You can send the data to a file using a `std::ofstream`, which is a special type of `std::ostream`.

Example

Output Streams

Input Streams

Quick test!

How familiar is this:

```
int x;
std::cin >> x;
```

Input Streams

Can only give you data.

- The `std::cin` stream is an example of an input stream.
- All input streams are of type `std::istream`.

Pull out data using stream extraction operator: `>>`

Extraction gets data from stream as a string and converts it into the appropriate type.

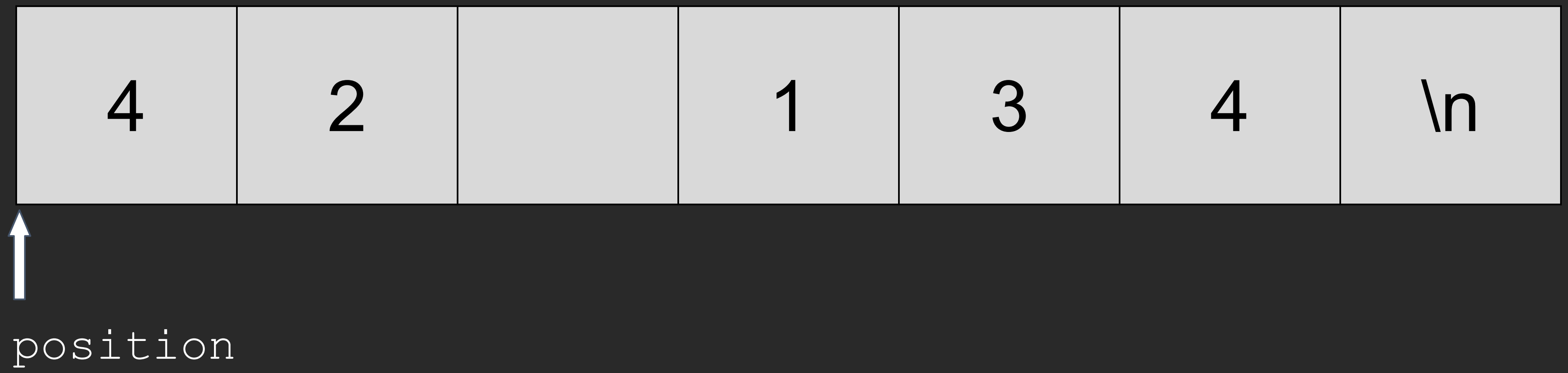
Input Streams

Just like with `std::ostream`, we can use a std::istream for more than just console IO.

You can read data from a file using a `std::ifstream`.

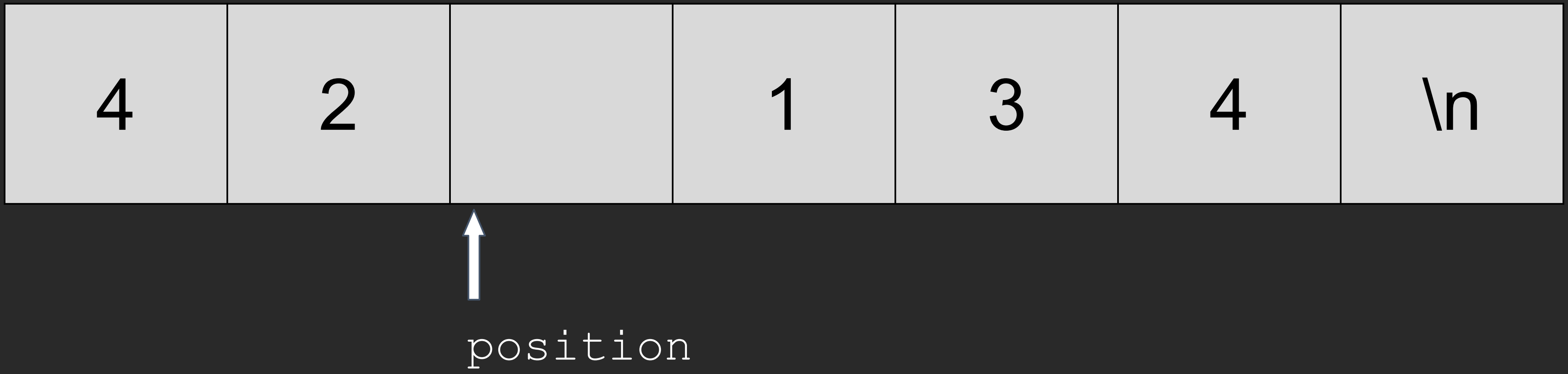
Input Streams

To understand a `std::istream`, think of it as a sequence of characters.



Input Streams

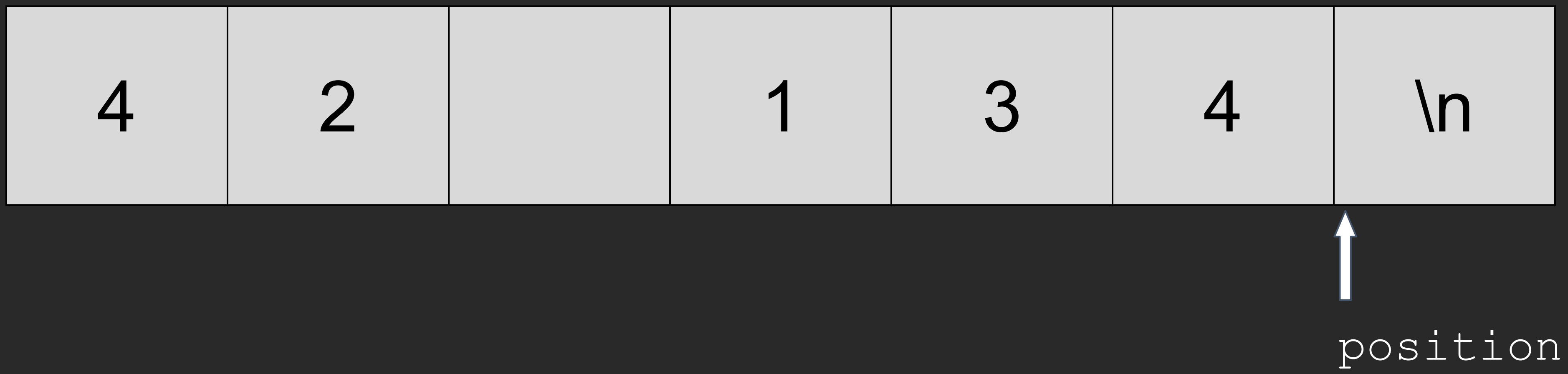
Extracting an integer will read as many characters as possible from the stream.



```
// input is an istream
int value;
input >> value;        // value is now 42
```

Input Streams

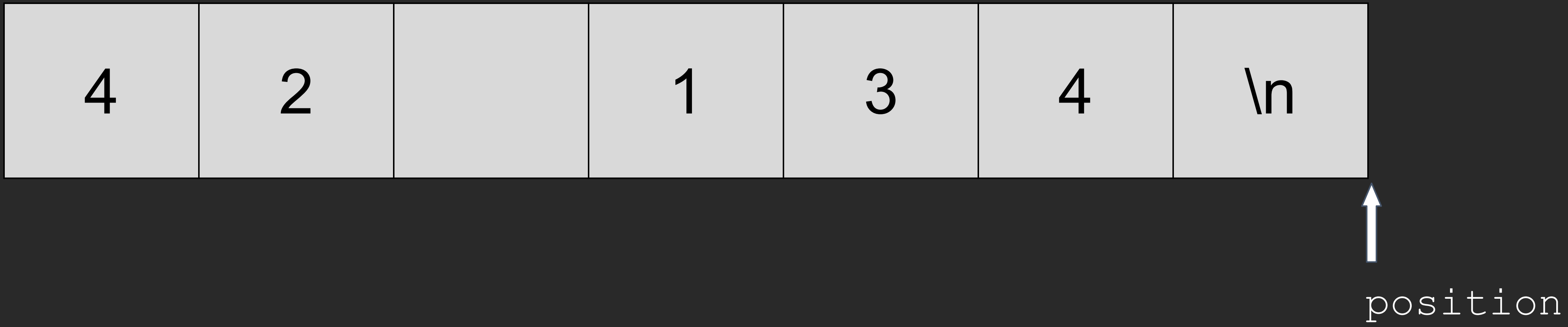
Extracting again will skip over any whitespace when reading the next integer.



```
// input is an istream
int value;
input >> value;      // value is now 134
```

Input Streams

When no more data is left, the fail bit will be set to true and `input.fail()` will return true.



```
// input is an istream
int value;
input >> value;      // value is now ??
```

Example

Input Streams

Reading Data From a File

There are some quirks with extracting a string from a stream.

Reading into a string using >> will only read a single word, not the whole line.

To read a whole line, use

```
getline(istream& stream, string& line);
```

Example

Input Streams using getline

Be careful about mixing >> with getline!

- >> reads up to the next whitespace character, and *does not* go past that whitespace character.
- getline reads up to the next delimiter (by default '\n'), and *does* go past ("consume") that delimiter.
- Generally only choose one or the other to read from your stream!

- Note: Do not use >> with the Stanford libraries which use getline.

Stream Miscellany

Some additional methods for using streams:

```
input.get(ch);                // reads one char at a time
input.clear();                // resets any fail bits
input.open("filename");       // open stream on file
input.seekg(0);               // rewinds stream to start

input.close();                // closes stream
                          // This is no longer necessary!
```

Some Questions to Ponder

What happens if you read into the wrong type?

Can you extract user defined types (e.g. classes) from a stream?

Can you control how output streams output the data we give them?

Is there a stream that might be both an input and output stream?

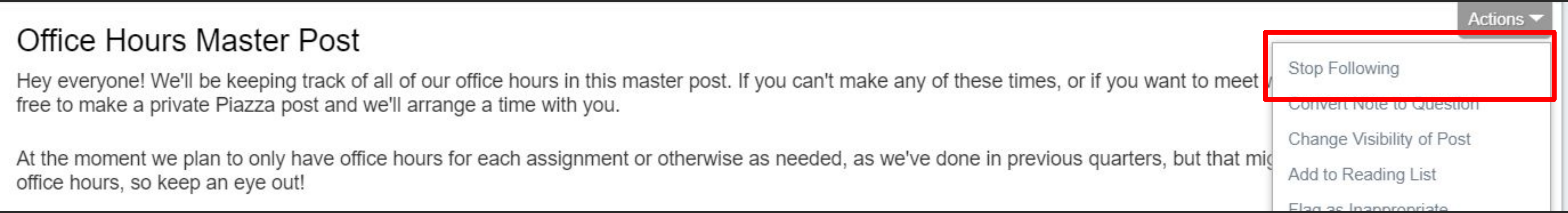
When does cin prompt the user for input?

Your turn: Answer in the Q&A!

Announcements

Logistics

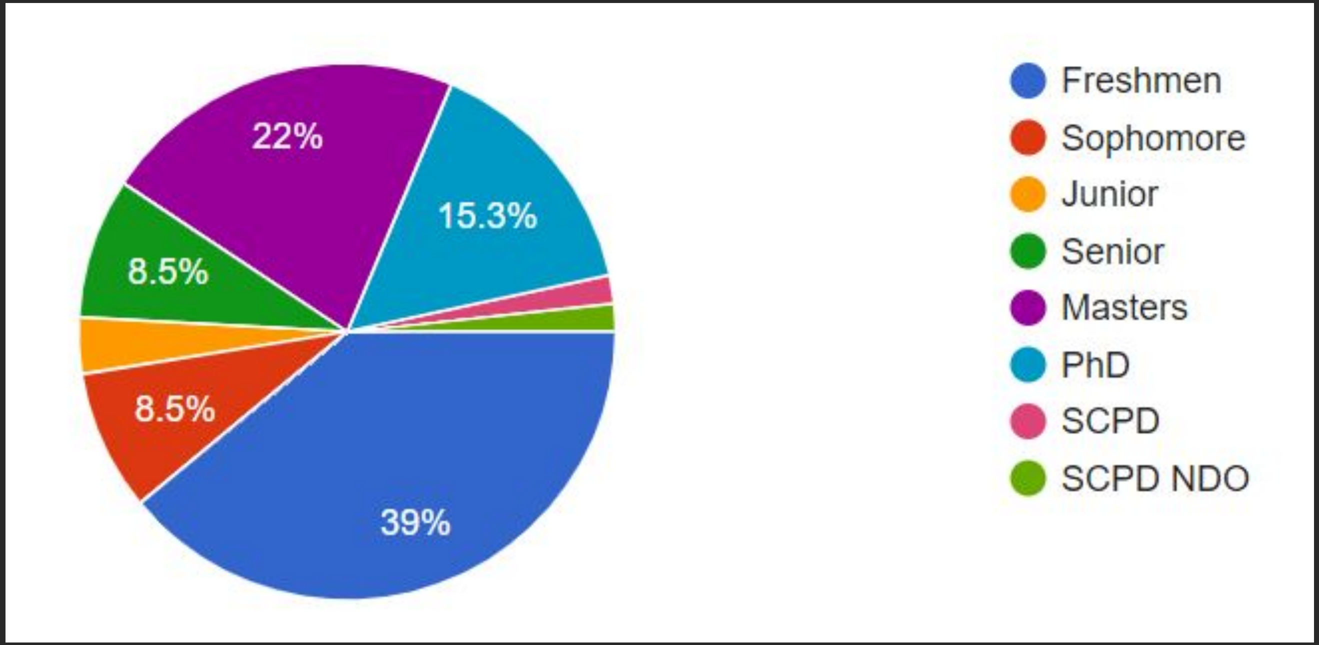
- Follow the Office Hours Master Post to get updates on office hours!


Office Hours Master Post

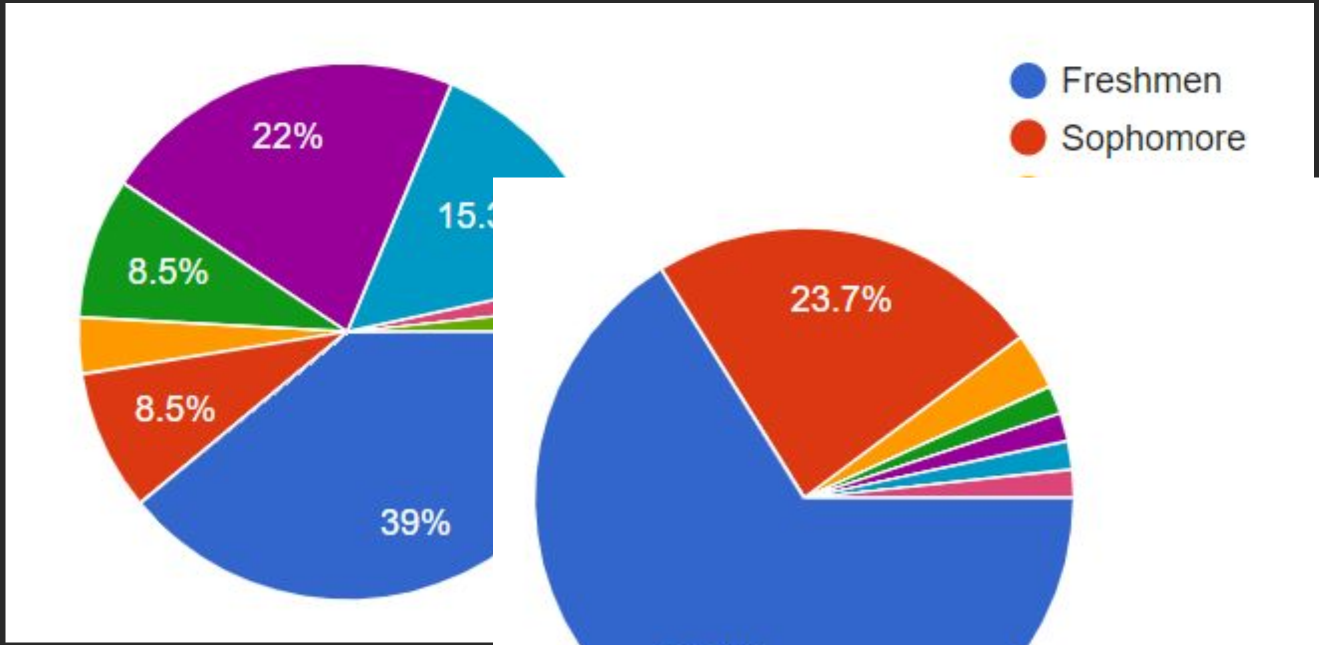
Hey everyone! We'll be keeping track of all of our office hours in this master post. If you can't make any of these times, or if you want to meet
free to make a private Piazza post and we'll arrange a time with you.

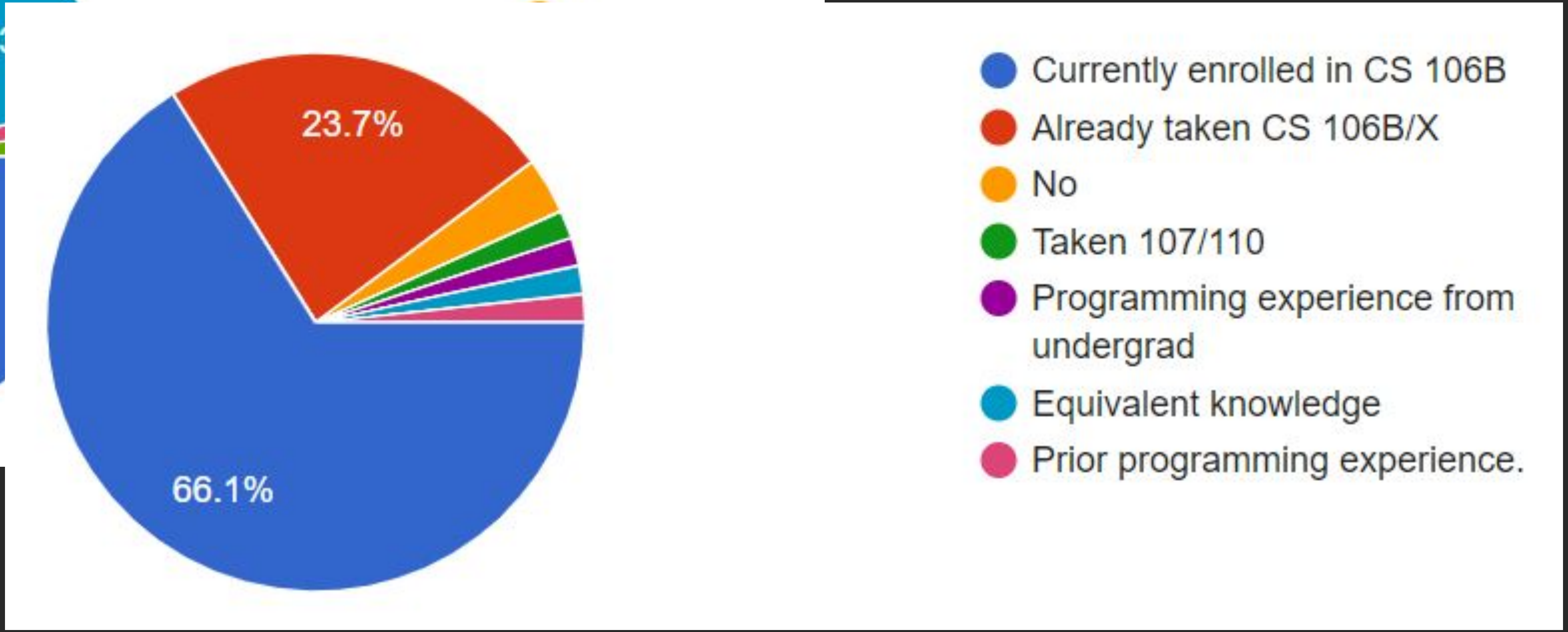
At the moment we plan to only have office hours for each assignment or otherwise as needed, as we've done in previous quarters, but that mi
office hours, so keep an eye out!

Survey Results!

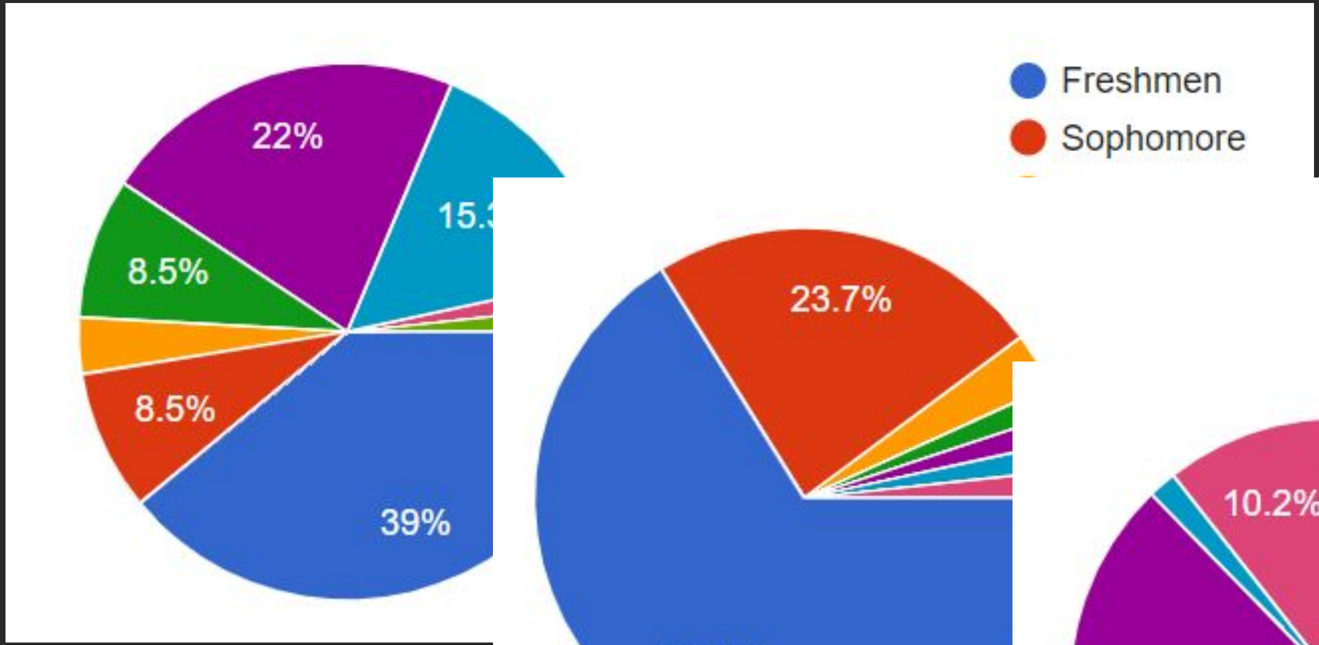


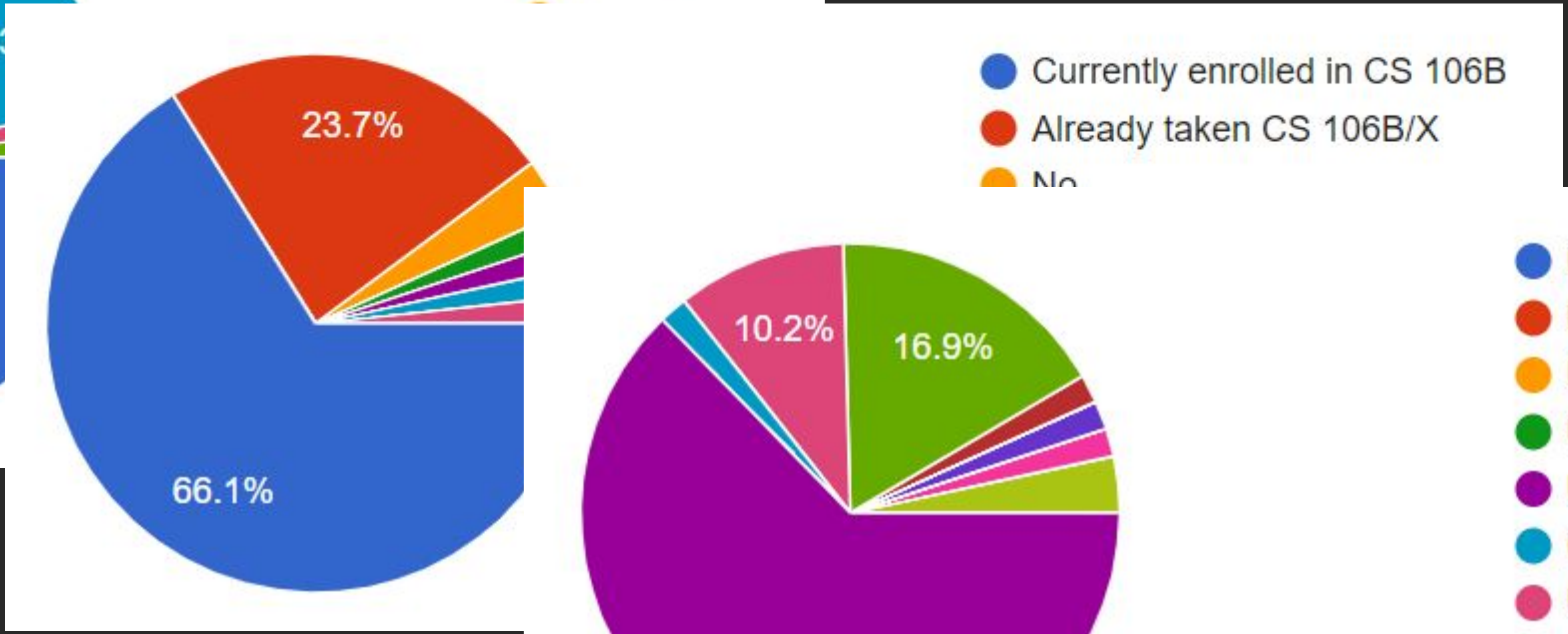
Survey Results!

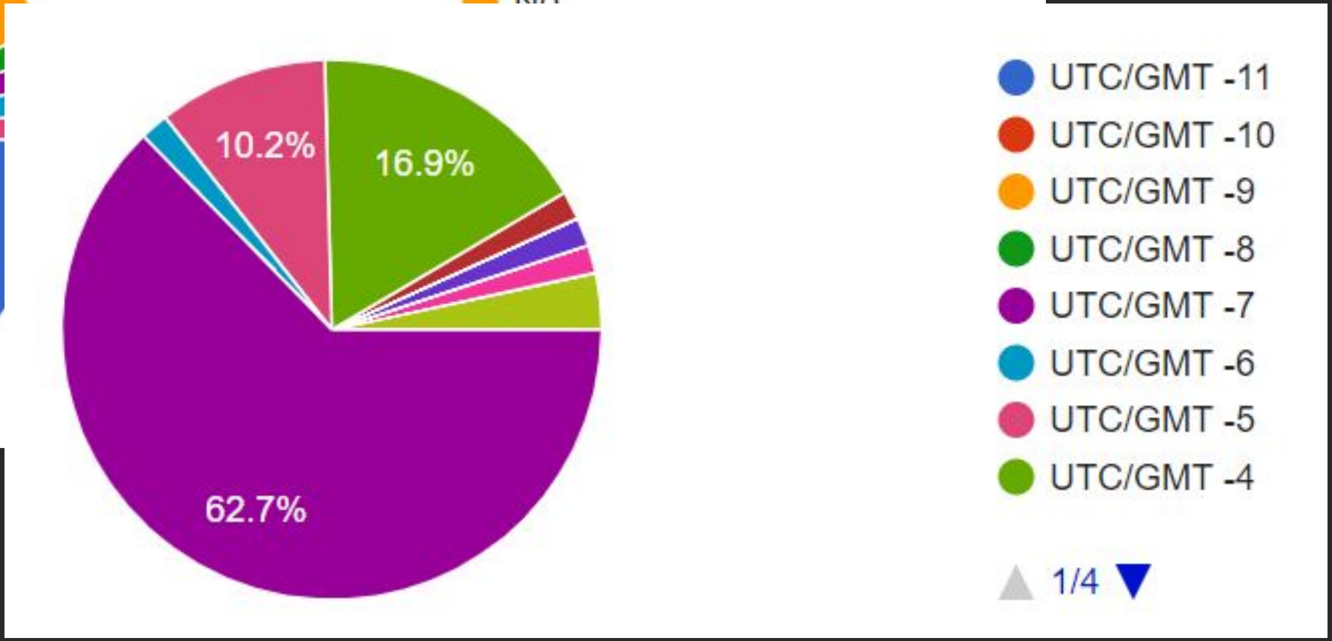




Survey Results!







Survey Results!

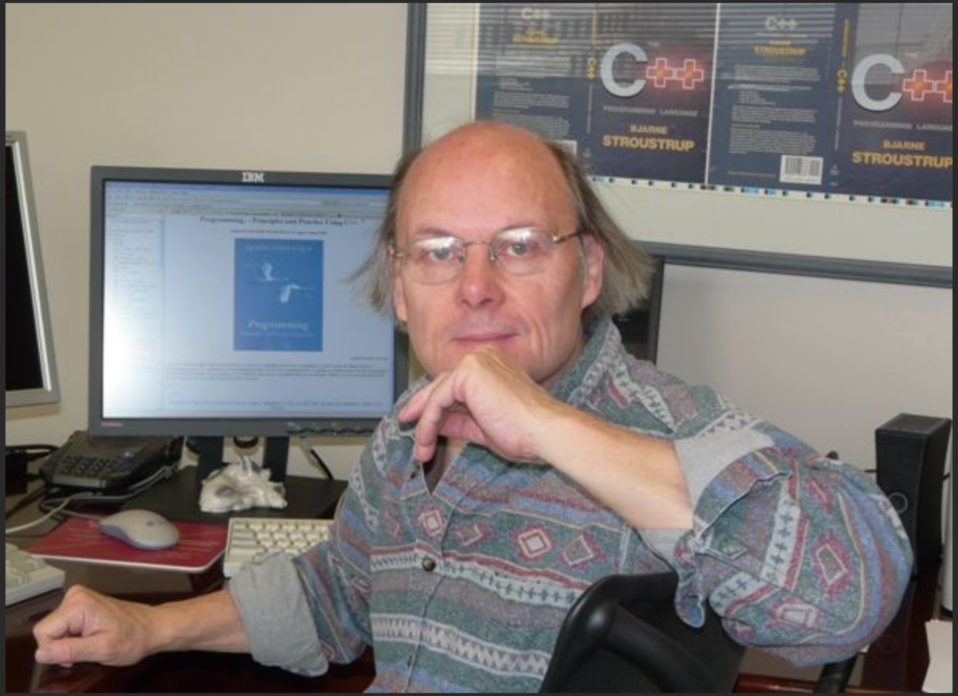
Majors/Programs:

- Aero/Astro
- Applied Physics
- Biology
- Biomedical Computation
- Chemistry
- Civil Engineering
- Computer Science
- Earth Systems
- Economics
- Electrical Engineering
- Energy Resources Engineering
- Engineering Physics
- Environmental Engineering
- Geoscience
- Japanese
- Journalism
- Management Science & Engineering
- Materials Science & Engineering
- Mathematics
- Mechanical Engineering
- Music Science and Technology
- Product Design
- Statistics
- Symbolic Systems
- Undecided :)

Survey Results!

Why you're here:

- C++ practice
- Industry usages
- Supplement CS 106B
- Personal projects
- Research
- Practice for coding interviews
- Modern C++ features
- Love of CS / C++! :)

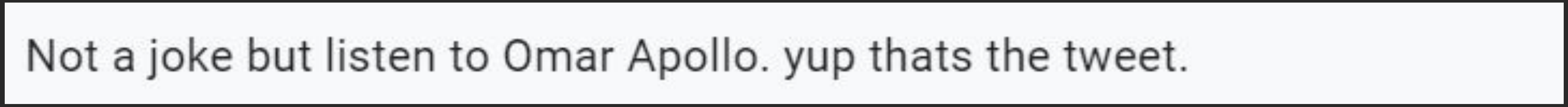


Bad Dad Joke of the Day:

- Two fish are in a tank. One says to the other: “Now, how do we drive this thing?”

Creds: Matthew

BONUS Bad Dad Joke:


Not a joke but listen to Omar Apollo. yup thats the tweet.

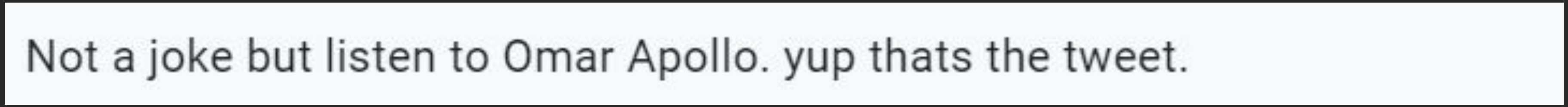
Creds: Xavier

Bad Dad Joke of the Day:

- Two fish are in a tank. One says to the other: “Now, how do we drive this thing?”

Creds: Matthew

BONUS Bad Dad Joke:


Not a joke but listen to Omar Apollo. yup thats the tweet.

Creds: Xavier

2-min stretch break!

Bad Dad Joke of the Day:

- Two fish are in a tank. One says to the other: “Now, how do we drive this thing?”

Creds: Matthew

Stream Internals

Buffering

Writing to a console/file is a slow operation.

If the program had to write each character immediately, runtime would significantly slow down.

What can we do?

Buffering

Idea:

Accumulate characters in a temporary buffer/array.

When buffer is full, write out all contents of the buffer to the output device at once.

This process is known as flushing the stream

Example

Stream Buffering

Buffering

The internal sequence of data stored in a stream is called a buffer.

Istreams use them to store data we haven’t used yet.

Ostreams use them to store data they haven’t passed along yet.

Question: There’s a third standard stream: `std::cerr`, the error stream, which is *not* buffered. Why might that be?

Flushing the Buffer

If we want to force the contents of the buffer to their destination, we can flush the stream:

```
stream << std::flush;     // use to print what you have
stream << std::endl;      // use if you want a newline
```

This is equivalent to: `stream << "\n" << std::flush;`

State Bits

Streams have four bits to indicate their state.

- G Good bit: ready for read/write.
- F Fail bit: previous operation failed, all future operations frozen.
- E EOF bit: previous operation reached the end of buffer content.
- B Bad bit: external error, likely irrecoverable.

Which bit to use?

1. Read data
2. Check if data is valid, if not break
3. Use data
4. Go back to step 1

```
while(true) {
    stream >> temp;
    if(stream.fail()) break; // checks for fail OR bad bit!
    do_something(temp);
}
```

Aside: Chaining >> or <<

Recall: >> and << are actually functions under the hood!

```
std::cout << "hello";
```

$\updownarrow$

```
operator<<(std::cout, "hello");
```

Aside: Chaining >> or <<

And we know that >> and << return a reference to the stream!

```
ostream& operator<<(ostream& os, const string& rhs);
// returns the stream passed in as os
```

This is why this works:

```
std::cout << "hello" << 23 << "world";
```

Aside: Chaining >> or <<

And we know that >> and << return a reference to the stream!

```
ostream& operator<<(ostream& os, const string& rhs);
// returns the stream passed in as os
```

This is why this works:

```
((std::cout << "hello") << 23) << "world";
```

Aside: Chaining >> or <<

And we know that >> and << return a reference to the stream!

```
ostream& operator<<(ostream& os, const string& rhs);
// returns the stream passed in as os
```

This is why this works:

```
((std::cout << "hello") << 23) << "world";
```

Aside: Chaining >> or <<

And we know that >> and << return a reference to the stream!

```
ostream& operator<<(ostream& os, const string& rhs);
// returns the stream passed in as os
```

This is why this works:

```
((std::cout) << 23) << "world";
```

Aside: Chaining >> or <<

And we know that >> and << return a reference to the stream!

```
ostream& operator<<(ostream& os, const string& rhs);
// returns the stream passed in as os
```

This is why this works:

```
(std::cout << 23) << "world";
```

Aside: Chaining >> or <<

And we know that >> and << return a reference to the stream!

```
ostream& operator<<(ostream& os, const string& rhs);
// returns the stream passed in as os
```

This is why this works:

```
(std::cout << 23) << "world";
```

Aside: Chaining >> or <<

And we know that >> and << return a reference to the stream!

```
ostream& operator<<(ostream& os, const string& rhs);
// returns the stream passed in as os
```

This is why this works:

```
(std::cout) << "world";
```

Aside: Chaining >> or <<

And we know that >> and << return a reference to the stream!

```
ostream& operator<<(ostream& os, const string& rhs);
// returns the stream passed in as os
```

This is why this works:

```
std::cout << "world";
```

Which bit to use? - Part 2

Let’s look at this code again:

```
while(true) {
    stream >> temp;
    if(stream.fail()) break;
    do_something(temp);
}
```

Which bit to use? - Part 2

Let’s look at this code again:

```
while(true) {
    stream >> temp;
    if(stream.fail()) break;
    do_something(temp);
}
```

Streams can be converted to bool

Which bit to use? - Part 2

Let’s look at this code again:

```
while(true) {
    stream >> temp;
    if (!stream) break;
    do_something(temp);
}
```

Streams can be converted to bool

Which bit to use? - Part 2

Let's look at this code again:

```
while(true) {
    stream >> temp;
    if (!stream) break; // !stream is an alias for stream.fail()
    do_something(temp);
}
```

Which bit to use? - Part 2

Let’s look at this code again:

We know this returns the stream.

```
while(true) {
    stream >> temp;
    if (!stream) break; // !stream is an alias for stream.fail()
    do_something(temp);
}
```

Which bit to use? - Part 2

Let’s look at this code again:

```
while(true) {
    if (!(stream >> temp)) break;
    do_something(temp);
}
```

Which bit to use? - Part 2

Let’s look at this code again:

```
while(true) {
    if (!(stream >> temp)) break;
    do_something(temp);
}
```

We can simplify the logic

Which bit to use? - Part 2

Let’s look at this code again:

```
while(stream >> temp) {
    do_something(temp);
}
```

Which bit to use? - Part 2

The same principle applies with getline:

```
while(stream >> temp) {
    do_something(temp);
}
```

```
while(getline(stream, temp)) {
    do_something(temp);
}
```

Stream Manipulators

Stream Manipulator

There are some special keywords that change the behaviour of the stream when inserted.

`std::endl` and `std::flush` are two examples.

Stream Manipulator

Common:

endl	inserts a newline and flushes the stream
ws	skips all whitespace until it finds another char
boolalpha	prints “true” and “false” for bools

Numeric:

boolalpha	prints “true” and “false” for bools
hex	prints numbers in hex
setprecision	adjusts the precision numbers print with

Padding:

setw	pads output
setfill	fills padding with character

Some examples - Padding

```
#include <iomanip>

std::cout << "["  << std::setw(10) << "Hi" << "]"
                                            <<  std::endl;
```

Outputs:

```
[        Hi]
```

Some examples - Padding

```
#include <iomanip>

std::cout << "["  << std::left
          << std::setw(10) << "Hi" << "]"  <<  std::endl;
```

Outputs:

```
[Hi        ]
```

Some examples - Padding

```
#include <iomanip>

std::cout << "["  << std::left << std::setfill('-')
            << std::setw(10) << "Hi" << "]"  <<  std::endl;
```

Outputs:

```
[Hi--------]
```

Some examples - Numeric

```
#include <iomanip>

std::cout << std::hex << 10;          // prints a
std::cout << std::oct << 10;          // prints 12
std::cout << std::dec << 10;          // prints 10
```

Stream Manipulators - Recap

Stream manipulators can be passed into streams to change how they behave.

They have a variety of uses, and if you'd like to format something differently, there's probably a manipulator for it.

You can find a list of the most common ones at http://www.cplusplus.com/reference/library/manipulators/

std::stringstream

stringstream

Sometimes we want to be able to treat a string like a stream.

Useful scenarios:

- Converting between data types
- Tokenizing a string

stringstream

```
#include <sstream>
std::string line = "137 2.718 Hello";
std::stringstream stream(line);

int myInt;
double myDouble;
std::string myString;
stream >> myInt >> myDouble >> myString;

std::cout << myInt     <<  std::endl;
std::cout << myDouble <<  std::endl;
std::cout << myString <<  std::endl;
```

When to use stringstream

- If you need error checking for user input, best practice is to:
    - use getline to retrieve a line from cin,
    - create a stringstream with the line,
    - parse the line using a stringstream, usually with >>.
- Use state bits to control streams and perform error-checking.
    - fail bit can check type mismatches
    - eof bit can check if you consumed all input

Next time

Start of the STL!
Containers