Class Design

Bad Dad Joke of the Day:

- person 1: knock knock
- dad: yellow

Creds: Simon

Game Plan

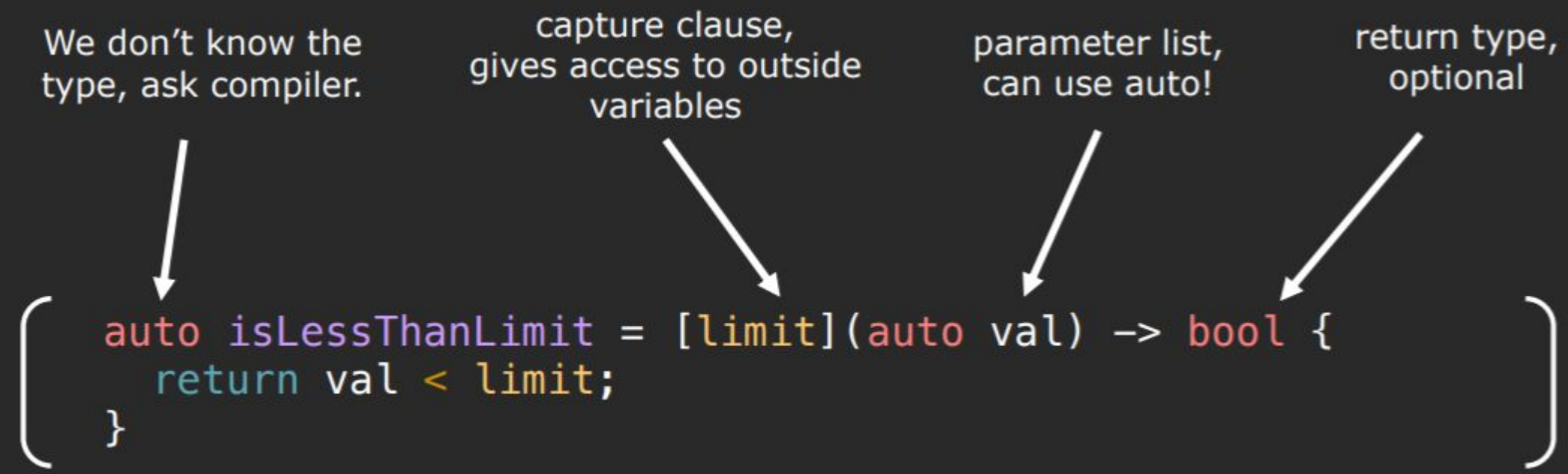
- Finishing up the BSTLR
- Announcements
- Brief Intro to Classes
- Everything `const`!

The BSTLR

1. Streams
2. Sequence containers + container adaptors
3. Associative containers
4. Iterators
5. Templates
6. Lambdas
7. Algorithms

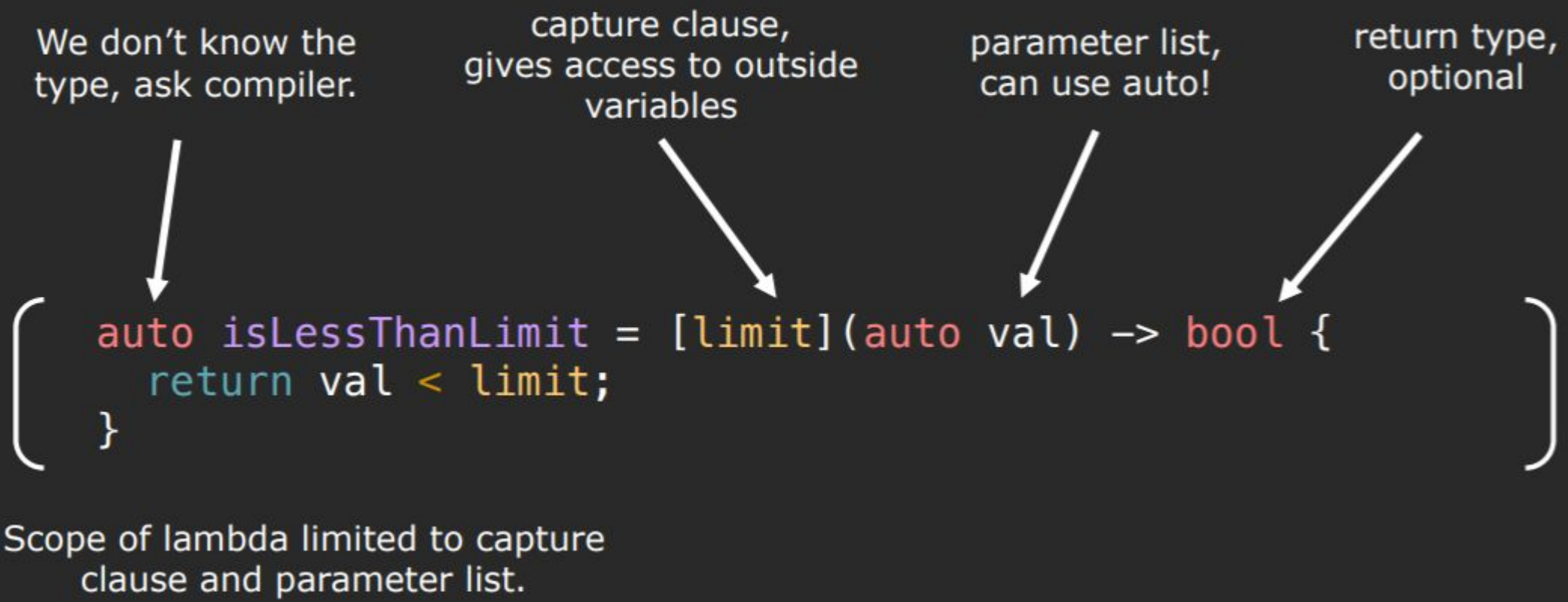
Lambdas Recap

Lambdas Recap



Scope of lambda limited to capture clause and parameter list.

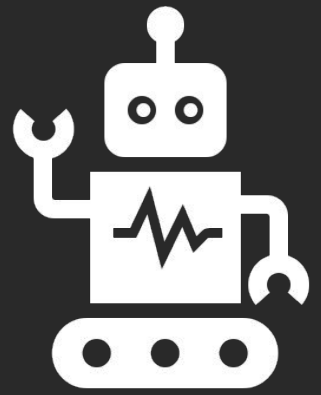
Lambdas Recap



Capture clause:

- [=, &obj] ➔ captures everything by value, except obj by reference
- [&, limit] ➔ captures everything by reference, except limit by value

Challenge #5: Lambdas



```
string fileToString(ifstream& file)
```

Algorithms Recap

Algorithms Recap

Algorithms we've seen:

- std::sort
- std::find
- std::count
- std::nth_element
- std::stable_partition
- std::copy
- std::copy_if
- std::remove_if
- and more!

Algorithms Recap

Algorithms we've seen:

- std::sort
- std::find
- std::count
- std::nth_element
- std::stable_partition
- std::copy
- std::copy_if
- std::remove_if
- and more!

Special iterators:

- back_inserter
  - e.g., std::copy(vec.begin(), vec.end(), std::back_inserter(newVec));
- stream_iterator
  - e.g., std::copy(vec.begin(), vec.end(), std::ostream_iterator<int>(cout, ", "));

Algorithms Recap

Algorithms we've seen:

- std::sort
- std::find
- std::count
- std::nth_element
- std::stable_partition
- std::copy
- std::copy_if
- std::remove_if
- and more!

Special iterators:

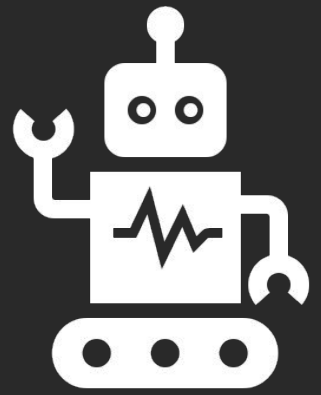
- back_inserter
  - e.g., std::copy(vec.begin(), vec.end(), std::back_inserter(newVec));
- stream_iterator
  - e.g., std::copy(vec.begin(), vec.end(), std::ostream_iterator<int>(cout, ", "));

Erase-remove idiom using algorithms*:

```
std::erase(
    std::remove(v.begin(), v.end()), v.end()
);
```

*many containers will define their own erase function which does this for you - this only applies if you use the STL erase/remove algorithms

Challenge #6: Algorithms



```
int dotProduct(const vector<int>& v1,
               const vector<int>& v2)
```

STL Wrap-Up:
Let's put it all together!

THE

FEDERALIST:

A COLLECTION OF

ESSAYS,

WRITTEN IN FAVOUR OF THE

NEW CONSTITUTION,

AS AGREED UPON BY THE

FEDERAL CONVENTION,

SEPTEMBER 17, 1787.

IN TWO VOLUMES.
VOL. I.

NEW-YORK:
PRINTED AND SOLD BY JOHN TIEBOUT,
No. 358 PEARL-STREET.
1799.

THE

FEDERALIST:

A COLLECTION OF

ESSAYS,

WRITTEN IN FAVOUR OF THE

NEW CONSTITUTION,

AS AGREED UPON BY THE

FEDERAL CONVENTION,

SEPTEMBER 17, 1787.

IN TWO VOLUMES.
VOL. I.

NEW-YORK:
PRINTED AND SOLD BY JOHN TIEBOUT,
No. 358 PEARL-STREET.
1799.

and good Type, in one handsome Volume duodecimo, and delivered to subscribers at the moderate price of one dollar. A few copies will be printed on superfine royal writing paper, price ten shillings.

No money required till delivery.

To render this work more complete, will be added, without any additional expence,

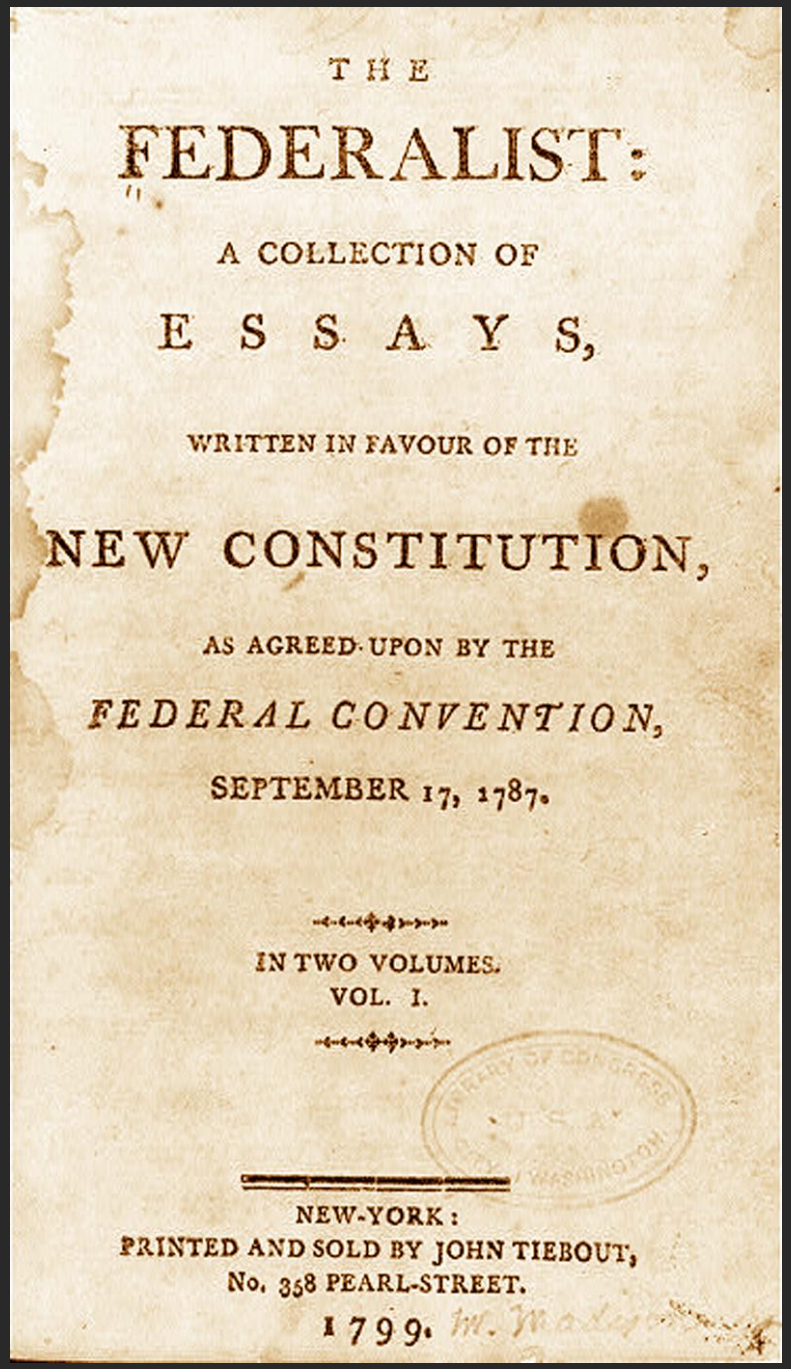
PHILO-PUBLIUS,

AND THE

Articles of the Convention,

As agreed upon at Philadelphia, September 17th, 1787.

IF ICESRC I



THE

FEDERALIST:

A COLLECTION OF

ESSAYS,

WRITTEN IN FAVOUR OF THE

NEW CONSTITUTION,

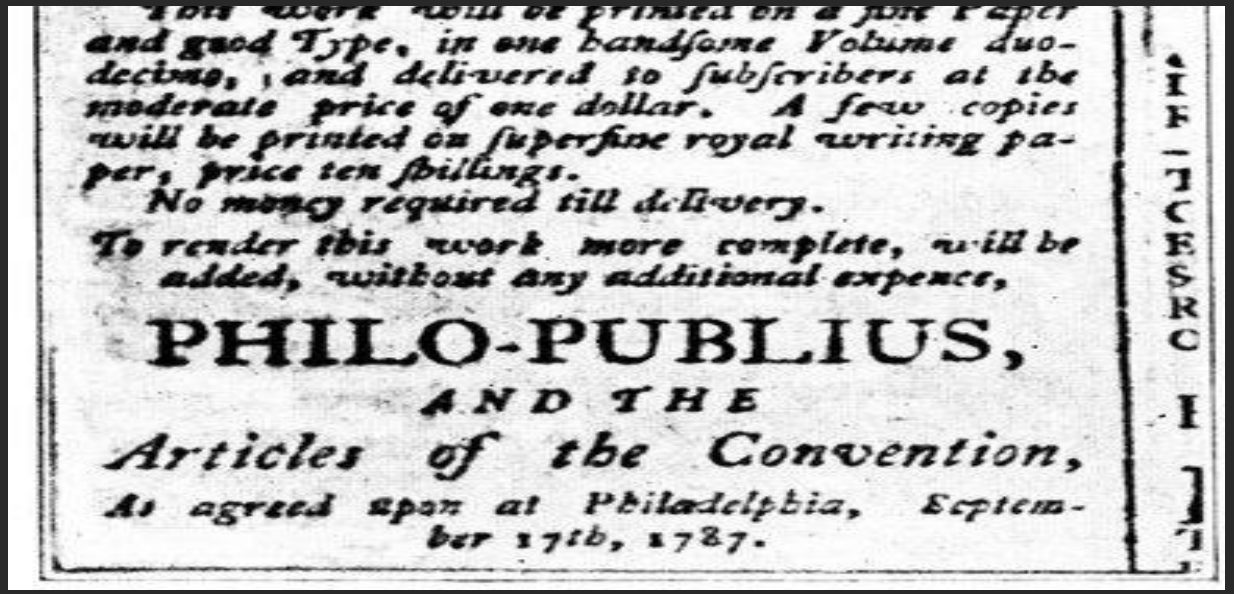
AS AGREED UPON BY THE

FEDERAL CONVENTION,

SEPTEMBER 17, 1787.

IN TWO VOLUMES.
VOL. I.

NEW-YORK:
PRINTED AND SOLD BY JOHN TIEBOUT,
No. 358 PEARL-STREET.
1799.



and good Type, in one handſome Volume duodecimo, and delivered to ſubſcribers at the moderate price of one dollar. A few copies will be printed on ſuperfine royal writing paper, price ten ſhillings.

No money required till delivery.

To render this work more complete, will be added, without any additional expence,

PHILO-PUBLIUS,

AND THE

Articles of the Convention,

As agreed upon at Philadelphia, September 17th, 1787.

The FŒDERALIST, No. 10.

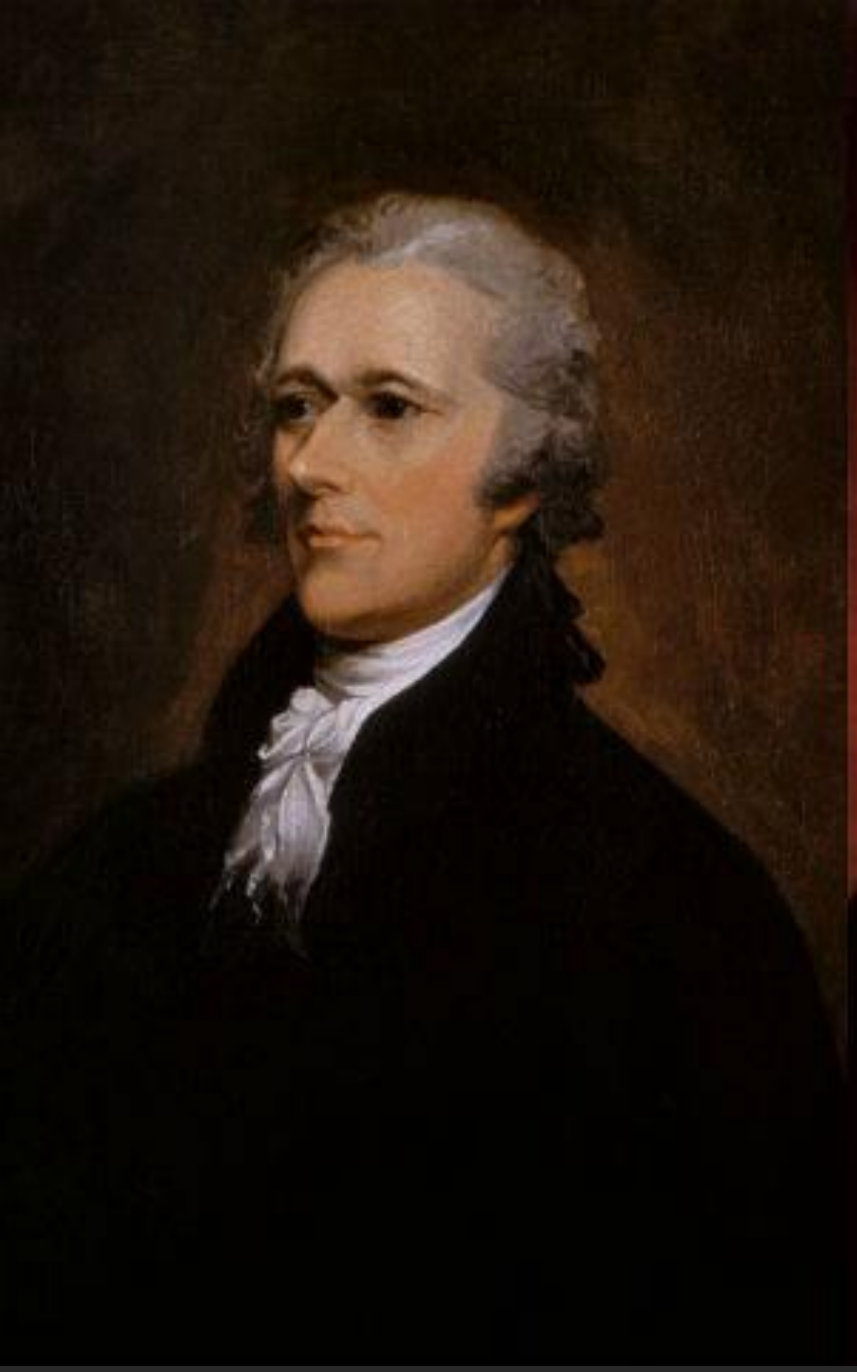
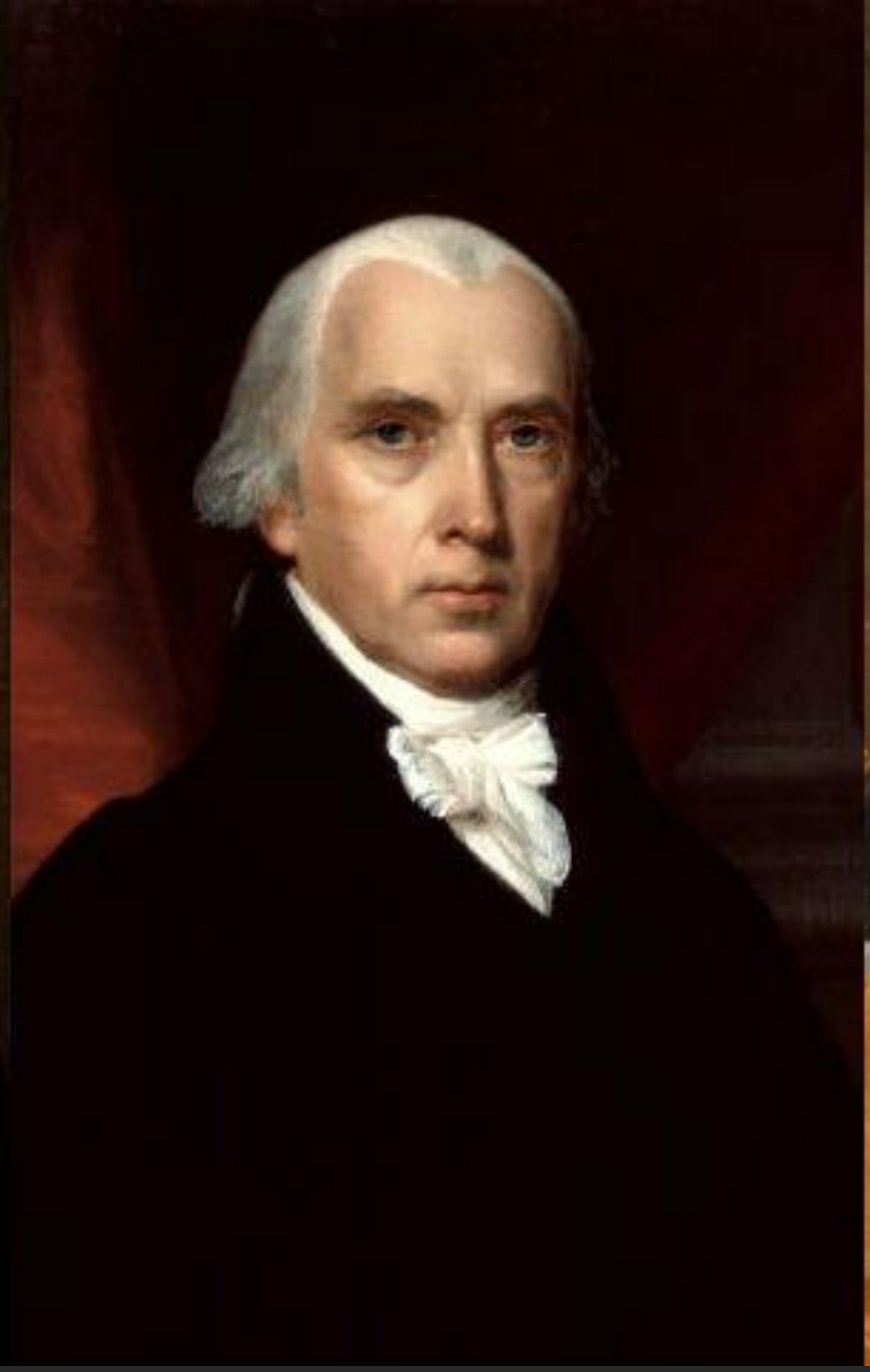
To the People of the State of New-York.

AMONG the numerous advantages promiſed by a well conſtructed Union, none deſerves to be more accurately developed than its tendency to break and control the violence of faction. The friend of popular governments, never finds himſelf ſo much alarmed for their character and fate, as when he contemplates their propenſity to this dangerous vice. He will not fail therefore to ſet a due value on any plan which, without violating the principles to which he is attached, provides a proper cure for it. The inſtability, injuſtice and confuſion introduced into the public councils, have in truth been the mortal diſeaſes under which popular governments have every where periſhed; as they continue to be the favorite and fruitful topics from which the adverſaries to liberty derive their moſt ſpecious declamations. The valuable improvements made by the American Conſtitutions on the popular models, both ancient and modern, cannot certainly

The influence of factious leaders may kindle a flame within their particular States, but will be unable to ſpread a general conflagration through the other States: A religious ſect, may degenerate into a political faction in a part of the confederacy; but the variety of ſects diſperſed over the entire face of it, muſt ſecure the national Councils againſt any danger from that ſource: A rage for paper money, for an abolition of debts, for an equal diviſion of property, or for any other improper or wicked project, will be leſs apt to pervade the whole body of the Union, than a particular member of it; in the ſame proportion as ſuch a malady is more likely to taint a particular county or diſtrict, than an entire State.

In the extent and proper ſtructure of the Union, therefore, we behold a republican remedy for the diſeaſes moſt incident to republican Government. And according to the degree of pleaſure and pride, we feel in being Republicans, ought to be our zeal in cheriſhing the ſpirit and ſupporting the character of Fœderaliſts.

PUBLIUS.


Can we discover an author’s identity from their writing?

Can we discover an author’s identity from their writing?

stylometry noun

sty·lom·e·try | \ stī ˈlämə·trē, -tri\

plural -es

Definition of *stylometry*

: the study of the chronology and development of an author's work based especially on the recurrence of particular turns of expression or trends of thought

The Idea

Authors have an underlying writing style.

Subconsciously writers tend to write in a consistent manner.

...

The Idea

Authors have an underlying writing style.

Subconsciously writers tend to write in a consistent manner.

...

Could we use these tendencies as a literary fingerprint?

The Idea

We need a writer invariant.

The Idea

We need a writer invariant.

Function words:

- Syntactic glue of a language
- E.g. *the, I, he, she, do, from, because...*

The Idea

Let’s imagine our language only has 3 function words:

```
[I, the, there]
```

Deep into that darkness peering, long I stood there, wondering, fearing, doubting, dreaming dreams no mortal ever dared to dream before.

- Edgar Allan Poe

I first met Dean not long after my wife and I split up. I had just gotten over a serious illness that I won’t bother to talk about, except that it had something to do with the miserably weary split-up and my feeling that everything there was dead.

- Jack Kerouac

The Idea

We can create a fingerprint vector for the two texts.

```
[I, the, there]
```

Deep into that darkness peering, long I stood there, wondering, fearing, doubting, dreaming dreams no mortal ever dared to dream before.

- Edgar Allan Poe

I first met Dean not long after my wife and I split up. I had just gotten over a serious illness that I won't bother to talk about, except that it had something to do with the miserably weary split-up and my feeling that everything there was dead.

- Jack Kerouac

The Idea

```
[I, the, there]

[ 0 ,  0 ,  0 ]
```

Deep into that darkness peering, long I stood there, wondering, fearing, doubting, dreaming dreams no mortal ever dared to dream before.

- Edgar Allan Poe

I first met Dean not long after my wife and I split up. I had just gotten over a serious illness that I won't bother to talk about, except that it had something to do with the miserably weary split-up and my feeling that everything there was dead.

- Jack Kerouac

The Idea

```
[I, the, there]

[ 0 ,  0 ,  0 ]
```

Deep into that darkness peering, long I stood there, wondering, fearing, doubting, dreaming dreams no mortal ever dared to dream before.

- Edgar Allan Poe

I first met Dean not long after my wife and I split up. I had just gotten over a serious illness that I won't bother to talk about, except that it had something to do with the miserably weary split-up and my feeling that everything there was dead.

- Jack Kerouac

The Idea

```
[I, the, there]

[ 0 ,  0 ,  0 ]
```

Deep into that darkness peering, long I stood there, wondering, fearing, doubting, dreaming dreams no mortal ever dared to dream before.

- Edgar Allan Poe

I first met Dean not long after my wife and I split up. I had just gotten over a serious illness that I won't bother to talk about, except that it had something to do with the miserably weary split-up and my feeling that everything there was dead.

- Jack Kerouac

The Idea

```
[I, the, there]

[ 1 ,  0 ,  0 ]
```

Deep into that darkness peering, long I stood there, wondering, fearing, doubting, dreaming dreams no mortal ever dared to dream before.

- Edgar Allan Poe

I first met Dean not long after my wife and I split up. I had just gotten over a serious illness that I won't bother to talk about, except that it had something to do with the miserably weary split-up and my feeling that everything there was dead.

- Jack Kerouac

The Idea

```
[I, the, there]

[ 1 ,  0 ,  0 ]
```

Deep into that darkness peering, long I stood there, wondering, fearing, doubting, dreaming dreams no mortal ever dared to dream before.

- Edgar Allan Poe

I first met Dean not long after my wife and I split up. I had just gotten over a serious illness that I won't bother to talk about, except that it had something to do with the miserably weary split-up and my feeling that everything there was dead.

- Jack Kerouac

The Idea

```
[I, the, there]

[ 1 ,  0 ,  0 ]
```

Deep into that darkness peering, long I stood **there**, wondering, fearing, doubting, dreaming dreams no mortal ever dared to dream before.

- Edgar Allan Poe

I first met Dean not long after my wife and I split up. I had just gotten over a serious illness that I won't bother to talk about, except that it had something to do with the miserably weary split-up and my feeling that everything there was dead.

- Jack Kerouac

The Idea

```
[I, the, there]

[ 1 ,  0 ,  0 ]
```

Deep into that darkness peering, long I stood **there**, wondering, fearing, doubting, dreaming dreams no mortal ever dared to dream before.

- Edgar Allan Poe

I first met Dean not long after my wife and I split up. I had just gotten over a serious illness that I won't bother to talk about, except that it had something to do with the miserably weary split-up and my feeling that everything there was dead.

- Jack Kerouac

The Idea

```
[I, the, there]

[ 1 ,  0 ,  1 ]
```

Deep into that darkness peering, long I stood there, wondering, fearing, doubting, dreaming dreams no mortal ever dared to dream before.

- Edgar Allan Poe

I first met Dean not long after my wife and I split up. I had just gotten over a serious illness that I won't bother to talk about, except that it had something to do with the miserably weary split-up and my feeling that everything there was dead.

- Jack Kerouac

The Idea

```
[I, the, there]
```

```
[ 1 ,  0 ,  1 ]
```

Deep into that darkness peering, long I stood there, wondering, fearing, doubting, dreaming dreams no mortal ever dared to dream before.

- Edgar Allan Poe

I first met Dean not long after my wife and I split up. I had just gotten over a serious illness that I won't bother to talk about, except that it had something to do with the miserably weary split-up and my feeling that everything there was dead.

- Jack Kerouac

The Idea

[I, the, there]

[1 , 0 , 1]

[0 , 0 , 0]

Deep into that darkness peering, long I stood there, wondering, fearing, doubting, dreaming dreams no mortal ever dared to dream before.

- Edgar Allan Poe

I first met Dean not long after my wife and I split up. I had just gotten over a serious illness that I won't bother to talk about, except that it had something to do with the miserably weary split-up and my feeling that everything there was dead.

- Jack Kerouac

The Idea

[I, the, there]

[1 , 0 , 1]

[0 , 0 , 0]

Deep into that darkness peering, long I stood there, wondering, fearing, doubting, dreaming dreams no mortal ever dared to dream before.

- Edgar Allan Poe

I first met Dean not long after my wife and I split up. I had just gotten over a serious illness that I won't bother to talk about, except that it had something to do with the miserably weary split-up and my feeling that everything there was dead.

- Jack Kerouac

The Idea

[I, the, there]

[1 , 0 , 1]

[0 , 0 , 0]

Deep into that darkness peering, long I stood there, wondering, fearing, doubting, dreaming dreams no mortal ever dared to dream before.

- Edgar Allan Poe

I first met Dean not long after my wife and I split up. I had just gotten over a serious illness that I won't bother to talk about, except that it had something to do with the miserably weary split-up and my feeling that everything there was dead.

- Jack Kerouac

The Idea

[I, the, there]

[1 , 0 , 1]

[4 , 0 , 0]

Deep into that darkness peering, long I stood there, wondering, fearing, doubting, dreaming dreams no mortal ever dared to dream before.

- Edgar Allan Poe

I first met Dean not long after my wife and I split up. I had just gotten over a serious illness that I won't bother to talk about, except that it had something to do with the miserably weary split-up and my feeling that everything there was dead.

- Jack Kerouac

The Idea

[I, **the**, there]

[1 , 0 , 1]

[4 , 0 , 0]

Deep into that darkness peering, long I stood there, wondering, fearing, doubting, dreaming dreams no mortal ever dared to dream before.

- Edgar Allan Poe

I first met Dean not long after my wife and I split up. I had just gotten over a serious illness that I won't bother to talk about, except that it had something to do with **the** miserably weary split-up and my feeling that everything there was dead.

- Jack Kerouac

The Idea

[I, **the**, there]

[1 , 0 , 1]

[4 , **0** , 0]

Deep into that darkness peering, long I stood there, wondering, fearing, doubting, dreaming dreams no mortal ever dared to dream before.

- Edgar Allan Poe

I first met Dean not long after my wife and I split up. I had just gotten over a serious illness that I won't bother to talk about, except that it had something to do with **the** miserably weary split-up and my feeling that everything there was dead.

- Jack Kerouac

The Idea

[I, **the**, there]

[1 , 0 , 1]

[4 , 1 , 0]

Deep into that darkness peering, long I stood there, wondering, fearing, doubting, dreaming dreams no mortal ever dared to dream before.

- Edgar Allan Poe

I first met Dean not long after my wife and I split up. I had just gotten over a serious illness that I won't bother to talk about, except that it had something to do with **the** miserably weary split-up and my feeling that everything there was dead.

- Jack Kerouac

The Idea

[I, the, **there**]

[1 , 0 , 1]

[4 , 1 , 0]

Deep into that darkness peering, long I stood there, wondering, fearing, doubting, dreaming dreams no mortal ever dared to dream before.

- Edgar Allan Poe

I first met Dean not long after my wife and I split up. I had just gotten over a serious illness that I won't bother to talk about, except that it had something to do with the miserably weary split-up and my feeling that everything **there** was dead.

- Jack Kerouac

The Idea

[I, the, **there**]

[1 , 0 , 1]

[4 , 1 , **0**]

Deep into that darkness peering, long I stood there, wondering, fearing, doubting, dreaming dreams no mortal ever dared to dream before.

- Edgar Allan Poe

I first met Dean not long after my wife and I split up. I had just gotten over a serious illness that I won't bother to talk about, except that it had something to do with the miserably weary split-up and my feeling that everything **there** was dead.

- Jack Kerouac

The Idea

[I, the, **there**]

[1 , 0 , 1]

[4 , 1 , 1]

Deep into that darkness peering, long I stood there, wondering, fearing, doubting, dreaming dreams no mortal ever dared to dream before.

- Edgar Allan Poe

I first met Dean not long after my wife and I split up. I had just gotten over a serious illness that I won't bother to talk about, except that it had something to do with the miserably weary split-up and my feeling that everything **there** was dead.

- Jack Kerouac

The Idea

```
[ 1 , 0 , 1 ]
```

Deep into that darkness peering, long I stood there, wondering, fearing, doubting, dreaming dreams no mortal ever dared to dream before.

- Edgar Allan Poe

```
[I, the, there]
```

```
[ 4 , 1 , 1 ]
```

I first met Dean not long after my wife and I split up. I had just gotten over a serious illness that I won't bother to talk about, except that it had something to do with the miserably weary split-up and my feeling that everything there was dead.

- Jack Kerouac

The Idea

```
[I, the, there]
```

```
[ 1 ,  0 ,  1 ]
```

Deep into that darkness peering, long I stood there, wondering, fearing, doubting, dreaming dreams no mortal ever dared to dream before.

- Edgar Allan Poe

```
[ 4 ,  1 ,  1 ]
```

I first met Dean not long after my wife and I split up. I had just gotten over a serious illness that I won't bother to talk about, except that it had something to do with the miserably weary split-up and my feeling that everything there was dead.

- Jack Kerouac

The Idea

```
[ 1 ,  0 ,  1 ]          [ 4 ,  1 ,  1 ]
```

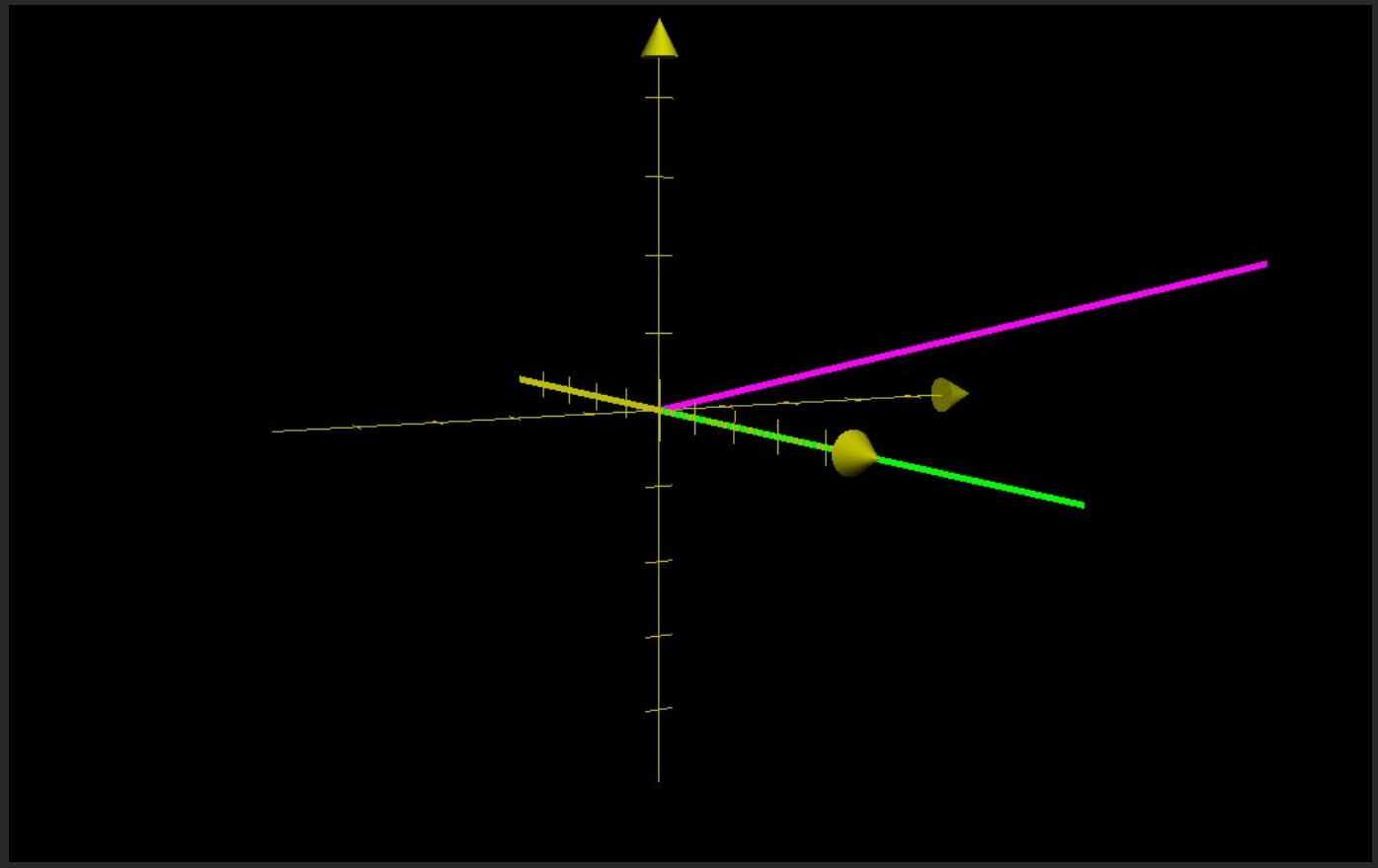
The Idea

```
[ 1 ,  0 ,  1 ]          [ 4 ,  1 ,  1 ]
```

The Idea

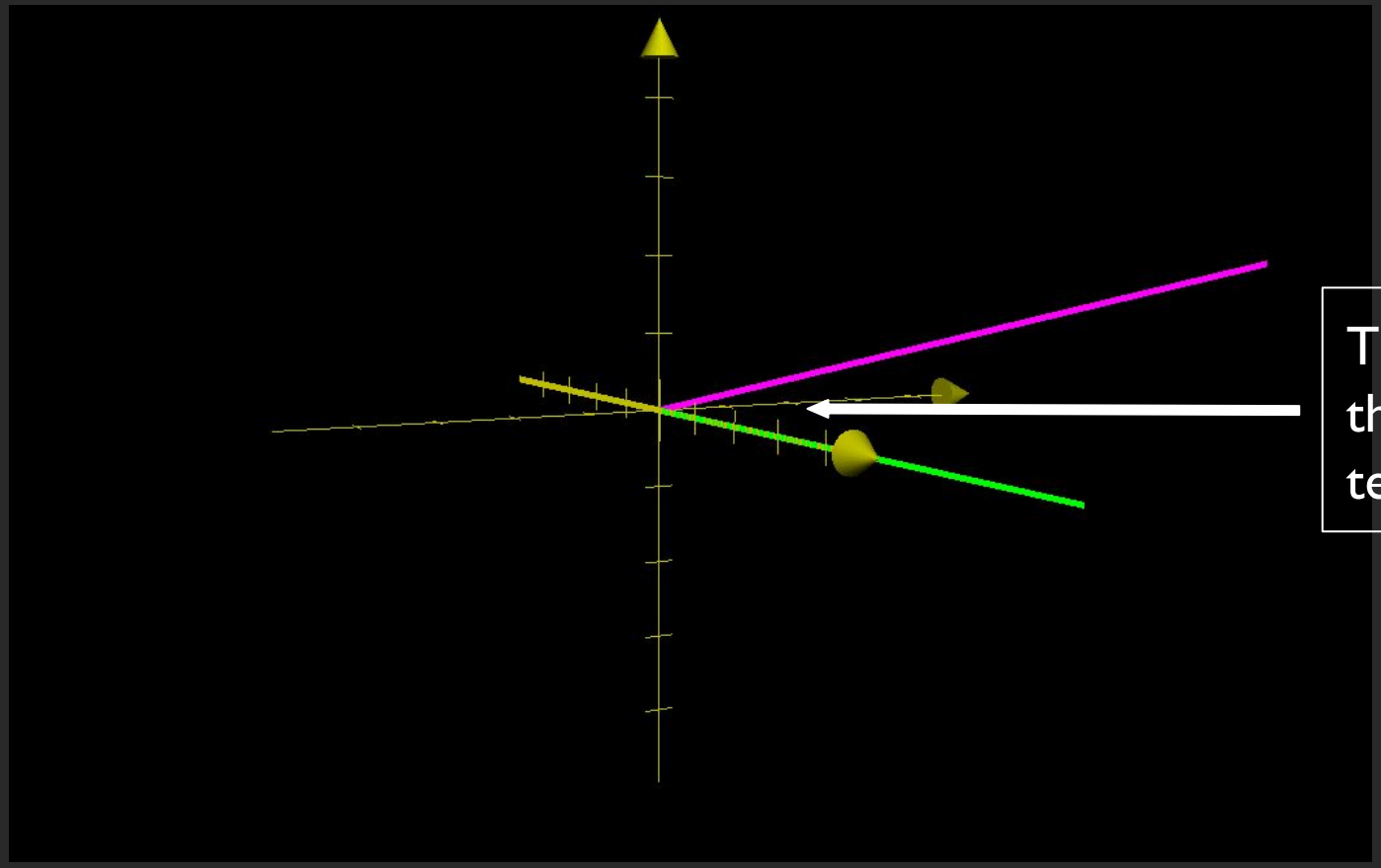
```
[ 1 ,  0 ,  1 ]
```

```
[ 4 ,  1 ,  1 ]
```



The Idea

`[ 1 , 0 , 1 ]` `[ 4 , 1 , 1 ]`

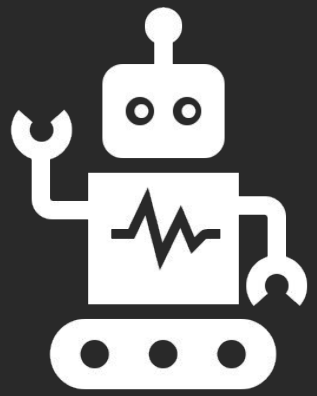


The closer this angle, the more similar the texts

The Idea

$$\cos\theta = \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\|\|\vec{v}\|}$$

Let’s get coding!



Example

Stylometry

Closing Notes

Closing Notes

The code for getting the word count (i.e. countOccurrences) will be really useful for the first part of assignment 2.

Congratulations!

"As mathematicians learned to lift theorems into their most general setting, so I wanted to lift algorithms and data structures."

— *Alex Stepanov, inventor of the STL*

Announcements

Announcements

- Please get your screenshots in for Assignment 2 by this Saturday, 11:59 pm!
- Assignment 2 will be due next Friday, 2/14
- Keep an eye on the Piazza for updated office hours!

Brief Intro to Classes

Starring Cynthia Lee's CS106B slides from last quarter!

Brief Intro to Classes

- Header files (.h) vs. source files (.cpp)
- Constructors
- Destructors
- Operator overloading
- Const

Header files (.h, .hh)...

Class declaration (.h)

Protection in case multiple .cpp files include this .h, so that its contents won't get declared twice

```
#ifndef _classname_h
#define _classname_h

class ClassName {
public:                              // in ClassName.h
    ClassName(parameters);           // constructor

    returnType name(parameters); // member functions
    returnType name(parameters); // (behavior inside
    returnType name(parameters); //  each object)

private:
    type _name;      // member variables
    type _name;      // (data inside each object)
};

#endif
```

IMPORTANT: must put a semicolon at end of class declaration

...vs. Source files (.cpp, .cc, etc.)

Member func. bodies

In ClassName.cpp, we write bodies (definitions) for the member functions that were declared in the .h file:

```
// ClassName.cpp
#include "ClassName.h"

// member function
returnType ClassName::methodName(parameters) {
    statements;
}
```

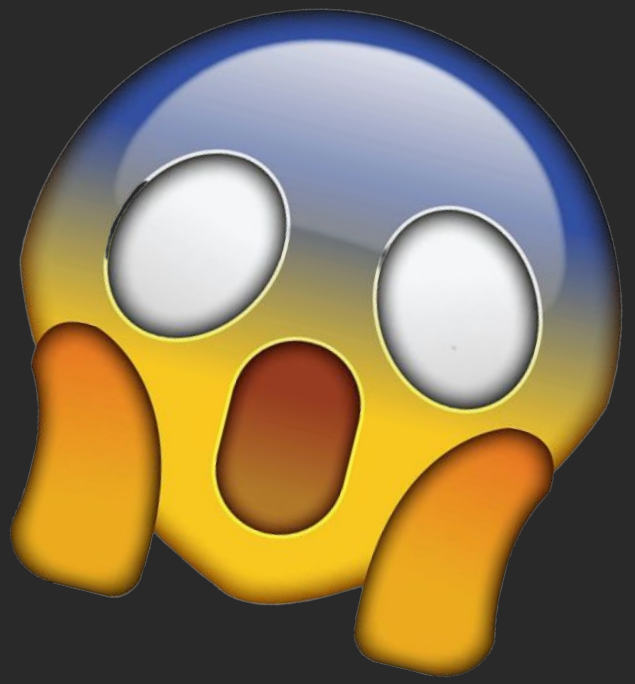
- Member functions/constructors can refer to the object's member variables.

Aside: Why so many extensions?

- Header file: .h, .hh, .hpp
- Source file: .cc, .cpp, .cxx, .c++, .C

Aside: Why so many extensions?

- Header file: .h, .hh, .hpp
- Source file: .cc, .cpp, .cxx, .c++, .C



Aside: Why so many extensions?

- Header file: .h, .hh, .hpp
- Source file: .cc, .cpp, .cxx, .c++, .C

- Depends on the compiler!*

Aside: Why so many extensions?

- Header file: .h, .hh, .hpp
- Source file: .cc, .cpp, .cxx, .c++, .C

- Depends on the compiler!*

- Historically, used .C (i.e. capital C)
- Now, Unix mostly uses .cc, and outside Unix mostly uses .cpp
- .h is technically for C programs, so if mixing C and C++ code, use .hh instead

*If interested, see the third answer under:
https://stackoverflow.com/questions/1545080/c-code-file-extension-cc-vs-cpp/1545085

Constructors

Constructors

Constructors

```
ClassName::ClassName(parameters) {
    statements to initialize the object;
}
```

Constructor: Initializes state of new objects as they are created.

- no return type is specified; implicitly "returns" the new object

- without constructor:

```
BankAccount ba;
ba._name = "Cynthia";
ba._balance = 1.25;                // tedious
```

- with constructor:

```
BankAccount ba("Cynthia", 1.25);    // better
```

Destructors

Destructors

Destructor (12.3)

```
// ClassName.h
~ClassName();
```

```
// ClassName.cpp
ClassName::~ClassName() { ...
```

Destructor: Called when the object is deleted by the program.

- (when the object falls out of {} scope)

- Useful if your object needs to free any memory as it dies.
  - delete any pointers stored as private members
  - delete[] any arrays stored as private members
  - *(we haven't learned about delete yet, that's next week!)*

Operator overloading

Operator overloading (6.2)

operator overloading: Redefining the behavior of a common operator in the C++ language.

unary:	+ - ++ -- * &
	! ~ new delete
binary:	+ - * / % += -=
	*= /= %= & | && ||
	^ == != < > <= >=
	<< >> = [] -> () ,

Syntax:

```
returnType operator op(parameters);      // in the .h file for the class

returnType operator op(parameters) {     // in the .cpp file for the class
    statements;
};
```

- For example, for two variables of type Foo, **a + b** will use the code you write in:

```
Foo operator +(Foo& a, Foo& b) {
    // function body
}
```

Stanford University

Operator overloading

Operator overloading (6.2)

unary: + - ++ -- * &
! ~ new delete

= -=
| && ||
<= >=
-> () ,

Next lecture!

Synt

retu

e class

retu

he class

};

- F

```
Foo operator +(Foo& a, Foo& b) {
    // function body
}
```

Const...

`Const...`

`...` Everything

Const Correctness

...

Credits to: Mike Precup
(with slight modifications)

Why Const?

"I still sometimes come across programmers who think const isn't worth the trouble. ‘Aw, const is a pain to write everywhere,’ I've heard some complain. ‘If I use it in one place, I have to use it all the time. And anyway, other people skip it, and their programs work fine. Some of the libraries that I use aren't const-correct either. Is const worth it?’

We could imagine a similar scene, this time at a rifle range: ‘Aw, this gun's safety is a pain to set all the time. And anyway, some other people don't use it either, and some of them haven't shot their own feet off...’

Safety-incorrect riflemen are not long for this world. Nor are const-incorrect programmers, carpenters who don't have time for hard-hats, and electricians who don't have time to identify the live wire. **There is no excuse for ignoring the safety mechanisms provided with a product, and there is particularly no excuse for programmers too lazy to write const-correct code**."

- Herb Sutter, generally cool dude

Why Const?

Instead of asking why you think **const** is important, I want to start with a different (related) question:

Why don't we use global variables?

Why Const?

- "Global variables can be read or modified by any part of the program, making it difficult to remember or reason about every possible use"
- "A global variable can be get or set by any part of the program, and any rules regarding its use can be easily broken or forgotten"

Why Const?

- "Non-const variables can be read or modified by any part of the function, making it difficult to remember or reason about every possible use"
- "A non-const variable can be get or set by any part of the function, and any rules regarding its use can be easily broken or forgotten"

Why Const?

Find the bug in this code:

```
void f(int x, int y) {
  if ((x==2 && y==3)||(x==1))
     cout << 'a' << endl;
  if ((y==x-1)&&(x==-1||y=-1))
     cout << 'b' << endl;
  if ((x==3)&&(y==2*x))
     cout << 'c' << endl;
}
```

Why Const?

Find the bug in this code:

```
void f(int x, int y) {
  if ((x==2 && y==3)||(x==1))
     cout << 'a' << endl;
  if ((y==x-1)&&(x==-1||y=-1))
     cout << 'b' << endl;
  if ((x==3)&&(y==2*x))
     cout << 'c' << endl;
}
```

Why Const?

Find the bug in this code:

```
void f(const int x, const int y) {
  if ((x==2 && y==3)||(x==1))
     cout << 'a' << endl;
  if ((y==x-1)&&(x==-1||y=-1))
     cout << 'b' << endl;
  if ((x==3)&&(y==2*x))
     cout << 'c' << endl;
}
```

Why Const?

The compiler finds the bug for us!

```
test.cpp: In function 'void f(int, int)':
test.cpp:7:31: error: assignment of read-only parameter 'y'
```

Why Const?

That's a fairly basic use case though, is that really all that const is good for?

Why Const?

No.

The Const Model

```
Planet earth;
```

The Const Model

```
int countPeople(Planet& p);
//...
int population = countPeople(earth);
```

The Const Model

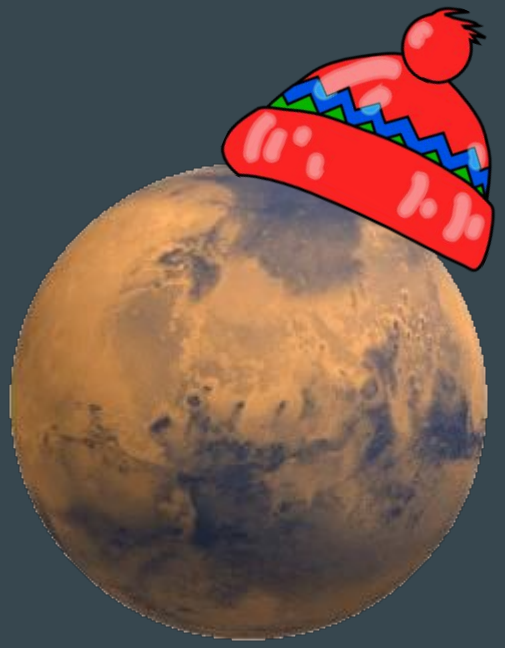
```
addLittleHat(earth);
```

countPeople(earth)

The Const Model

```
marsify(earth);
```

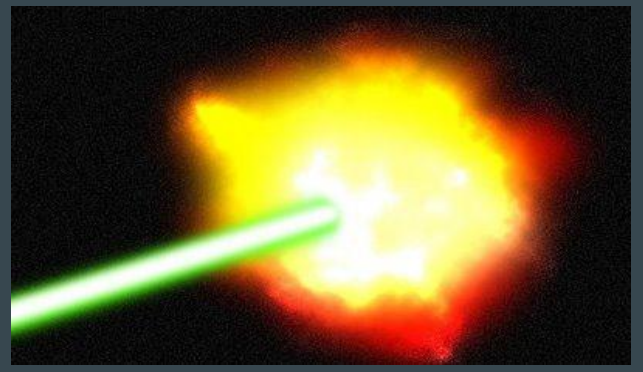
countPeople(earth)



The Const Model

```
deathStar(earth);
```

countPeople(earth)



Why Const?

How did this happen?

The Const Model

```
long int countPeople(Planet& p) {
  // Hats are the cornerstone of modern society
  addLittleHat(p);

  // More land; oceans were wasting space
  marsify(p);

  // Optimization: destroy planet
  // This makes population counting O(1)
  deathStar(p);
  return 0;
}
```

The Const Model

What would happen if I made that a const method?

The Const Model

```
long int countPopulation(const Planet& p) {
  // Hats are the cornerstone of modern society
  addLittleHat(p);

  // More land; oceans were wasting space
  marsify(p);

  // Optimization: destroy planet
  // This makes population counting O(1)
  deathStar(p);
  return 0;
}
```

The Const Model

```
test.cpp: In function ‘long int countPopulation(const Planet&)’:

test.cpp:9:21: error: invalid initialization of reference of type
‘Planet&’ from expression of type ‘const Planet’
test.cpp:3:6: error: in passing argument 1 of ‘void
addLittleHat(Planet&)’

test.cpp:12:12: error: invalid initialization of reference of type
‘Planet&’ from expression of type ‘const Planet’
test.cpp:4:6: error: in passing argument 1 of ‘void marsify(Planet&)’

test.cpp:16:14: error: invalid initialization of reference of type
‘Planet&’ from expression of type ‘const Planet’
test.cpp:5:6: error: in passing argument 1 of ‘void deathStar(Planet&)’
```

The Const Model

const allows us to reason about whether a variable will be changed.

useful for clients to use properly, the written class is used by others!

The Const Model

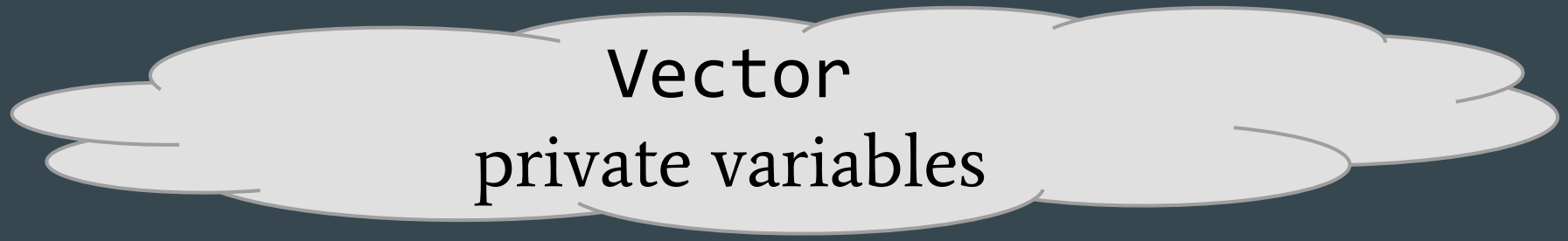
```
void f(int& x) {
  // The value of x here
  aConstMethod(x);
  anotherConstMethod(x);
  // Is the same value of x here
}
```

const and Classes

How does `const` interact with classes?

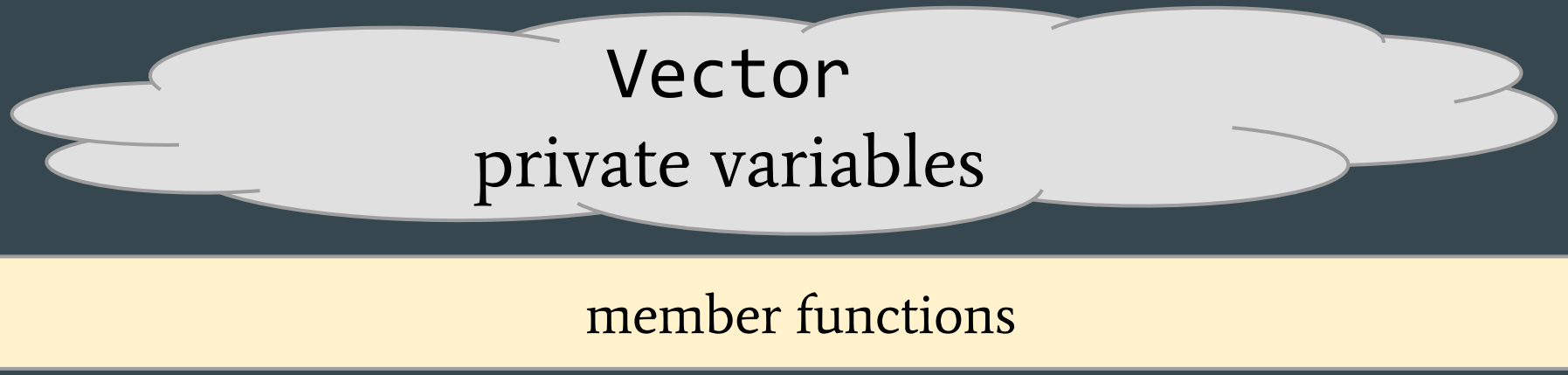
How do we define `const` member functions?

const and Classes



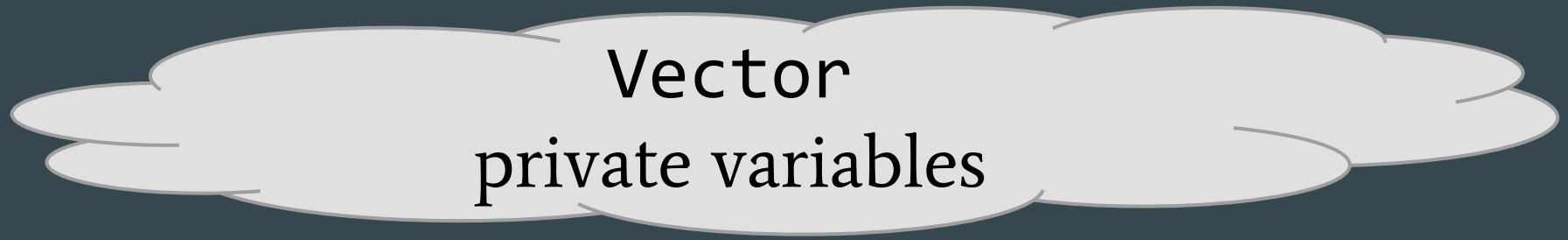
Let's have this cloud represent the member variables of the Vector class

const and Classes



Previously, we thought that you just used member functions to interact with an instance of a vector

const and Classes



Now we see that there are both const and non-const member functions, and const objects can't use non-const member functions

const variable cannot be passed into an non-const function

The Const Model

```
// Defining const member functions
struct Planet {
   int countPopulation() const;
   void deathStar();
};

int Planet::countPopulation() const {
   return 42; // seems about right
}

void Planet::deathStar() {
   cout << "BOOM" << endl;
}
```

The Const Model

```
// Using const member functions
struct Planet {
   int countPopulation() const;
   void deathStar();
};

void evil(const Planet &p) {
   // OK: countPopulation is const
   cout << p.countPopulation() << endl;
   // ERROR: deathStar isn't const
   p.deathStar();
}
```

p is a const object, const object cannot use non-const functions

A Const Pointer

- Using pointers with const is a little tricky
  - When in doubt, read right to left

```
//constant pointer to a non-constant int
int * const p;   // (*p)++; OK!

                 // p++; NOT allowed!
```

A Const Pointer

- Using pointers with const is a little tricky
  - When in doubt, read right to left

* and const, read from right(*,const) to left orderly

```
//constant pointer to a non-constant int
int * const p;

//non-constant pointer to a constant int
const int* p;
```

A Const Pointer

- Using pointers with const is a little tricky
  - When in doubt, read right to left

```
//constant pointer to a non-constant int
int * const p;

//non-constant pointer to a constant int
const int* p;
int const* p;
```

A Const Pointer

- Using pointers with const is a little tricky
  - When in doubt, read right to left

```
//constant pointer to a non-constant int
int * const p;

//non-constant pointer to a constant int
const int* p;
int const* p;

//constant pointer to a constant int
const int* const p;
int const* const p;
```

A Const Pointer

- Using pointers with const is a little tricky
  - When in doubt, read right to left

```
//constant pointer to a non-constant Widget
Widget * const p;

//non-constant pointer to a constant Widget
const Widget* p;
Widget const* p;

//constant pointer to a constant Widget
const Widget* const p;
Widget const* const p;
```

Const Iterators

- Remember that iterators act like pointers
- `const vector<int>::iterator itr` however, acts like `int* const itr`
- To make an iterator read only, define a new `const_iterator`

```
vector v{1,2312};

const vector<int>::iterator itr  =  v.begin();

++itr;   // doesnt compile

*itr = 15; // compiles
```

Const Iterators

```
const vector<int>::iterator itr = v.begin();
*itr = 5; //OK! changing what itr points to
++itr; //ERROR! can’t modify itr
```

```
vector<int>::const_iterator itr = v.begin();
*itr = 5; //ERROR! can’t change value of itr
++itr; //OK! changing v
int value = *itr; //OK! reading from itr
```

Recap

Where does const work? type, functions, ptr, objects

It can be used as a **qualifier** on any type. This works for everything from arguments to local variables to return values.

```
const string &s = f();
```

It can also be used on functions:

```
size_t Vector<ElemType>::size() const;
```

Challenge Mode:

```
const int* const myClassMethod(const int* const & param) const;
```

Recap

- For the most part, always anything that does not get modified should be marked const
- Pass by const reference is better than pass by value
  - Not true for primitives (`bool`, `int`, etc)
- Member functions should have both const and non const iterators
- Read right to left to understand pointers
- Please don’t make a method to blow up earth

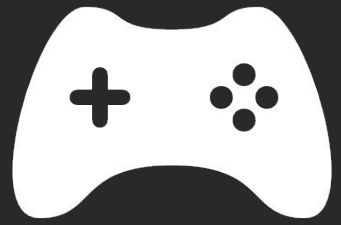
Final Notes

`const` on objects:

> Guarantees that the object won't change by allowing you to call only const functions and treating all public members as if they were const. This helps the programmer write safe code, and also gives the compiler more information to use to optimize.

`const` on functions:

> Guarantees that the function won't call anything but const functions, and won't modify any non-static, non-mutable members.



Next time

Operators