Bad Dad Joke of the Day:
- “Why was the ocean so salty?”
- “Because the sun didn’t wave back hahahahahahaha”

Creds: MH
Game Plan

- Finishing up Stylometry
- Announcements
- Review of Classes
- Everything \texttt{const}!
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. 338 PEARL- STREET.
1799.

This work will be printed on fine paper
and good type, in one handsome volume doubled,
and delivered to subscribers at the moderate price of one dollar. A few copies
will be printed on superfine royal writing pa-
paper, price ten shillings.
No money required till delivery.
To render this work more complete, will be
added, without any additional expense,
PHILO-PUBLIUS,
AND THE ARTICLES OF THE CONVENTION,
As agreed upon at Philadelphia, September 17
1787.

The FÆDERALIST, 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 dan-
gerous vice. He will not fail therefore to let a due
value on any plan which, without violating the
principles to which he is attached, provides a pro-
cure for it. The infallibility, injustice and con-
fusion 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, are chiefly

The influence of fœdacious leaders may kindle a
flame within their particular States, but will be un-
able 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 fields 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 pro-
jects, 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 beheld a republican remedy for the
diseases most incident to republican Government.
And according to the degree of pleasure and pro-
fit we feel in being Republicans, ought to be our zeal in
cherishing the Spirit and supporting the character of
Federallists.

PUBLIUS.
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

\[ [1, 0, 0] \quad \quad \quad \quad \quad \quad [4, 1, 1] \]
The Idea

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

● Reminder that Assignment 2 Part A is due this Saturday!
  ○ Clarification: We will only be looking at the screenshots by the deadline. The coding part will not be graded until the Part B deadline.
  ○ Again, should not take more than 2 hours!

● Assignment 2 Part B will be released after this lecture! Due Thurs, Nov. 14
Object-Oriented Programming in C++
Object-Oriented Programming in C++

Operator overloading, Virtual functions

Template Classes

Inheritance
Object-Oriented Programming in C++

- Operator overloading,
- Virtual functions
- Template Classes
- Inheritance

OOPs Concepts:
- Polymorphism
- Inheritance
- Abstraction
- Encapsulation
- Class
- Object
Review of Classes from CS106B
Classes from CS106B

- Oct. 23 Lecture: Classes and Objects:
  - [http://web.stanford.edu/class/cs106b/lectures/14/14-lecture.pdf](http://web.stanford.edu/class/cs106b/lectures/14/14-lecture.pdf)

- Cynthia covered...
  - .h vs .cpp
  - Basic:
    - Constructors, destructors
    - Const
    - Operator overloading
Classes from CS106B

- Oct. 23 Lecture: Classes and Objects:
  - [http://web.stanford.edu/class/cs106b/lectures/14/14-lecture.pdf](http://web.stanford.edu/class/cs106b/lectures/14/14-lecture.pdf)

- Cynthia covered...
  - .h vs .cpp
  - Basic:
    - Constructors, destructors
    - Const
    - Operator overloading
Header files (.h, .hh)...

**Class declaration (.h)**

```cpp
#ifndef _classname_h
#define _classname_h

class ClassName {
    public:
        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
```

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

**IMPORTANT:** must put a semicolon at end of class declaration
Member func. bodies

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

```cpp
// 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**

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

- with constructor:
  ```cpp
  BankAccount ba("Cynthia", 1.25); // better
  ```
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!)*
Const keyword
**The keyword const**

Just like a const reference parameter can't be modified by the function:

```cpp
void foo(const BankAccount& ba) {
}
```

A const member function can't change the object's state:

```cpp
class BankAccount {
    ... 
    double getBalance() const;
};
```
**Const keyword**

The keyword `const`  
Just like a const reference parameter can't be modified by the function:

```c
void foo(const BankAccount& ba) {
```

A const member function can't change the object's state:

```c
class BankAccount { ...
    double getBalance() const;
```
The keyword const

Just like a const reference parameter can't be modified by the function:

```cpp
void foo(const BankAccount& ba) {
```

A const member function can't change the object's state:

```cpp
class BankAccount {
    ... double getBalance() const;
```
Operator overloading (6.2)

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

Syntax:

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

returnType operator op(parameters) {
    statements;
}
```

- For example, for two variables of type Foo, `a + b` will use the code you write in:
  ```cpp
  Foo operator +(Foo& a, Foo& b) {
      // function body
  }
  ```

Unary: `+ - ++ -- * & ! ~ new delete`

Binary: `+ - * / % += -= *= /= %= & | && || ^ == != < > <= >= << >> = [] -> () ,`
Operator overloading

Operator overloading (6.2)

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

Leading operator:
-= -= -= += += +=
| && ||
<= >=
-= () ,

Precedence:

next lecture!
Const...
Const...

... Everything
Const Correctness

Mike Precup (mprecup@stanford.edu)
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
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:

```cpp
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:

```cpp
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:

```cpp
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

```cpp
int countPeople(Planet& p);
//...
int population = countPeople(earth);
```
The Const Model

```java
countPeople(earth);
addLittleHat(earth);
```
The Const Model

```javascript
marsify(earth);
```

```
countPeople(earth)
```
The Const Model

countPeople(earth);

deadStar(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?
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.
The Const Model

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

This is great for things like ints, but 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 a certain Object.
Previously, we thought that you just used member functions to interact with an instance of an object
Now we see that there are both const and non-const member functions, and const objects can't use non-const member functions.
const and Classes

Object
Internal State

const interface

void foo(const Object& input);

non-const interface

void bar(Object& input);
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;
    // NOT OK: deathStar isn't const
    p.deathStar();
}
A Const Pointer

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

```c
//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

```cpp
//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

```c
//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

```c
//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

```cpp
//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`

```cpp
vector v{1, 2312};

const vector<int>::iterator itr = v.begin();
++itr;  // doesn't compile

*itr = 15;  // compiles
```
Const Iterators

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

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

Where does const work?

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

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

It can also be used on functions:

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