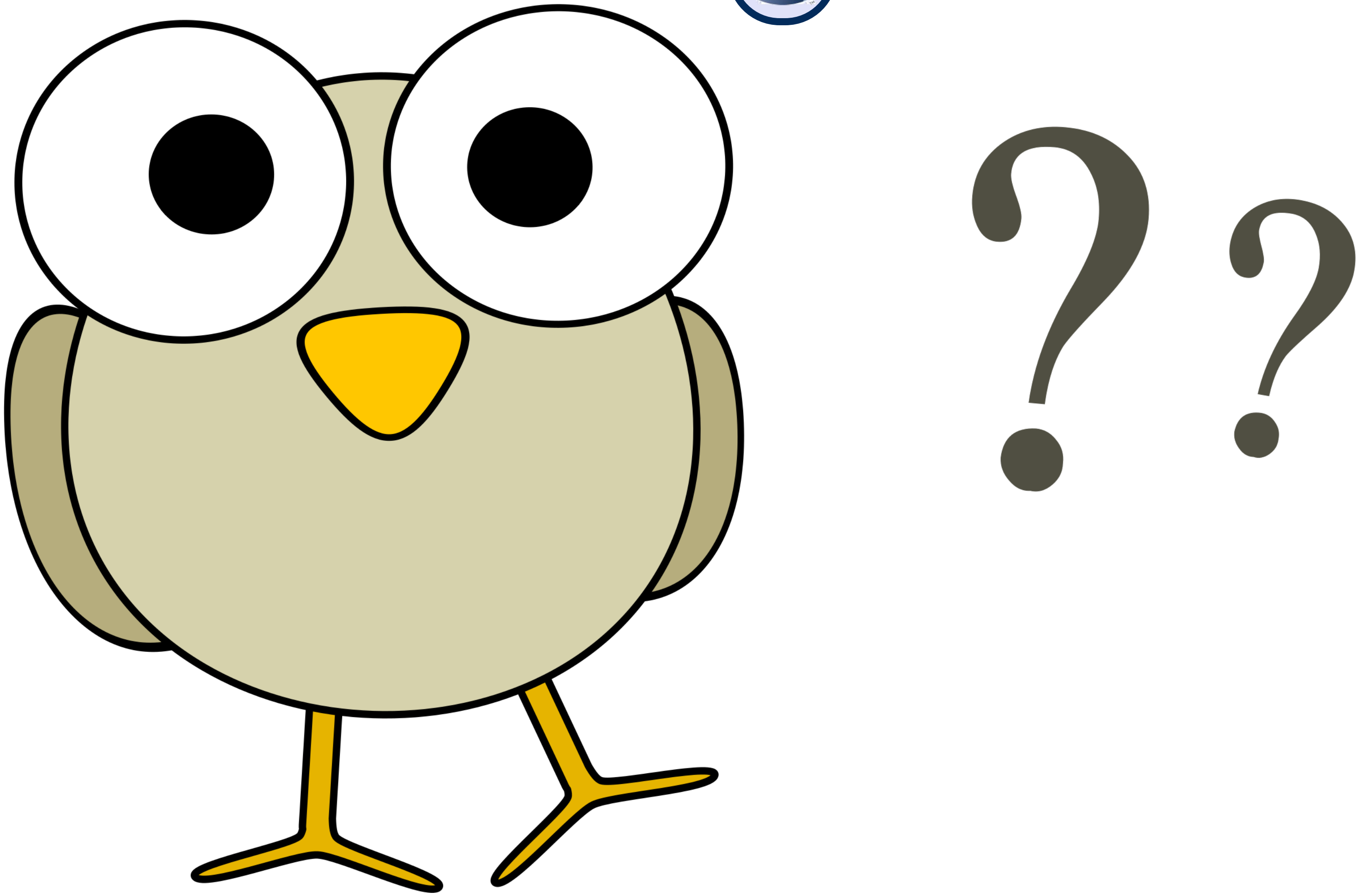
CSCI 200: Foundational Programming Concepts & Design Lecture 20

`const`

Previously in CSCI 200

- Collections of objects
- Functions with object parameters

Questions?



Learning Outcomes For Today

- Explain the four uses of const in our programs and classes.

On Tap For Today

- `const`
- Practice

On Tap For Today

- `const`

- Practice

Four times/ways to **const**

1. Variable modifier
2. Parameter modifier
3. Pointer modifier
4. Function modifier

Variable Modifier

- "Computer, **`PI`** is read-only. Don't let me change its value."

```
const double PI_D = 3.14159265;
float const PI_F = 3.141f;
```

Parameter Modifier

- "Computer, the **`print`** function should not modify the **`VALUE`**."

```
void print( const int VALUE ) {
    cout << VALUE;
}
```

Pointer Modifier

- “Computer, the **`pPI`** pointer should not modify the value it is pointing at.”

```
const float * pPI = new float(3.14f);
```

- “Computer, the **`P_pi`** pointer should not point at a different address.”

```
float * const P_pi = new float(3.14f);
```

- “Computer, the **`P_PI`** pointer should not point at a different address nor modify the value it is pointing at.”

```
const float * const P_PI = new float(3.14f);
```

Functions With `const`

- Can apply to parameter or return type

```
const float* generatePiPrecision(const int NUM_DECIMALS) {
  if(NUM_DECIMALS == 0) {
    return new float(3.f);
  } else if(NUM_DECIMALS == 1) {
    return new float(3.1f);
  } else if(NUM_DECIMALS == 2) {
    return new float(3.14f);
  } else {
    return new float(22.f / 7.f); // sure, approximate it poorly
  }
}

// and used later on

const float * const P_PI = generatePiPrecision(2);
```

Functions With `const`

- Can apply to parameter or return type

```
float* circle_area(const float * const P_PI, const float R) {
  return new float( *P_PI * R * R);
}


const float * const P_PI = generatePiPrecision(2);

float * pCircleArea = circle_area( P_PI, 2.0f )
```

Function Modifier

- "Computer, the **`getName`** function should NOT modify the callee."

```
// Zombie.h
class Zombie {
  public:
    Zombie(std::string);
    std::string getName() const;
  private:
    std::string _name;
};


// Zombie.cpp
string Zombie::getName() const {
    return _name;
}
```

Object Terminology

```
string Zombie::getName() const {
  return _name;
}
void Zombie::greet( const Zombie * pOTHER ) const {
  cout << "Hello " << pOTHER->getName() << endl;
  cout << "I am " << getName() << endl;
}
```

Who's name?

```
Zombie bill( "Bill" );
Zombie ted( "Ted" );
bill.greet( &ted );
```

Object Terminology

```
string Zombie::getName() const {
  return _name;
}
void Zombie::greet( const Zombie * pOTHER ) const {
  cout << "Hello " << pOTHER->getName() << endl;
  cout << "I am " << getName() << endl;
}

Zombie bill( "Bill" );
Zombie ted( "Ted" );
bill.greet( &ted );
```

Who's name?

The object upon which a function is called.

Object Terminology

```
string Zombie::getName() const {
  return _name;
}
void Zombie::greet( const Zombie * pOTHER ) const {
  cout << "Hello " << pOTHER->getName() << endl;
  cout << "I am " << getName() << endl;
}

Zombie bill( "Bill" );
Zombie ted( "Ted" );
bill.greet( &ted );
```

The callee's name

Object Terminology

```
string Zombie::getName() const {
  return _name;
}
void Zombie::greet( const Zombie * pOTHER ) const {
  cout << "Hello " << pOTHER->getName() << endl;
  cout << "I am " << getName() << endl;
}

Zombie bill( "Bill" );
Zombie ted( "Ted" );
bill.greet( &ted );
```

Who's name?

Object Terminology

```
string Zombie::getName() const {
  return _name;
}
void Zombie::greet( const Zombie * pOTHER ) const {
  cout << "Hello " << pOTHER->getName() << endl;
  cout << "I am " << getName() << endl;
}

Zombie bill( "Bill" );
Zombie ted( "Ted" );
bill.greet( &ted );
```

Who's name?

The object passed

Object Terminology

```
string Zombie::getName() const {
  return _name;
}
void Zombie::greet( const Zombie * pOTHER ) const {
  cout << "Hello " << pOTHER->getName() << endl;
  cout << "I am " << getName() << endl;
}

Zombie bill( "Bill" );
Zombie ted( "Ted" );
bill.greet( &ted );
```

The target's name

Object Terminology

```
string Zombie::getName() const {
  return _name;
}
void Zombie::greet( const Zombie * pOTHER ) const {
  cout << "Hello " << pOTHER->getName() << endl;
  cout << "I am " << getName() << endl;
}
```

The callee's name

The target's name

```
Zombie bill( "Bill" );
Zombie ted( "Ted" );
bill.greet( &ted );
```

Note…

```
string Zombie::getName() const {
  return _name;
}
void Zombie::greet( const Zombie * pOTHER ) const {
  cout << "Hello " << pOTHER->getName() << endl;
  cout << "I am " << getName() << endl;
}

Zombie bill( "Bill" );
Zombie ted( "Ted" );
bill.greet( &ted );
```

Note…

```
string Zombie::getName() {
  return _name;
}
void Zombie::greet( const Zombie * pOTHER ) const {
  cout << "Hello " << pOTHER->getName() << endl;
  cout << "I am " << getName() << endl;
}

Zombie bill( "Bill" );
Zombie ted( "Ted" );
bill.greet( &ted );
```

This won’t compile

Note…

```
string Zombie::getName() {
  return _name;
}
void Zombie::greet( const Zombie * pOTHER ) const {
  cout << "Hello " << pOTHER->getName() << endl;
  cout << "I am " << getName() << endl;
}


Zombie bill( "Bill" );
Zombie ted( "Ted" );
bill.greet( &ted );
```

const objects can only call const functions

Note…

```
string Zombie::getName() {
  return _name;
}
void Zombie::greet( const Zombie * pOTHER ) const {
  cout << "Hello " << pOTHER->getName() << endl;
  cout << "I am " << getName() << endl;
}


Zombie bill( "Bill" );
Zombie ted( "Ted" );
bill.greet( &ted );
```

const functions can only call other const functions

Note…

```
string Zombie::getName() const {
  return _name;
}
void Zombie::greet( const Zombie * pOTHER ) const {
  cout << "Hello " << pOTHER->getName() << endl;
  cout << "I am " << getName() << endl;
}

Zombie bill( "Bill" );
Zombie ted( "Ted" );
bill.greet( &ted );
```

Much better ☺

Note…#2

```
string Zombie::getName() const {
  return _name;
}
void Zombie::greet( const Zombie * pOTHER ) const {
  cout << "Hello " << pOTHER->getName() << endl;
  cout << "I am " << this->getName() << endl;
}

Zombie bill( "Bill" );
Zombie ted( "Ted" );
bill.greet( &ted );
```

`this`

- Returns a pointer to ourself

- `this` returns the address we are stored at
- *`this` returns us
  - Dereference the pointer to ourselves…references ourself

Function Modifier

- Which class functions should be marked as `const`?

On Tap For Today

- `const`

- Practice

To Do For Next Time

- Set3 due Tuesday
- Final Project Proposal due Wednesday

File I/O & Collections Quiz

- Make Canvas Full Screen
- Access Code:
- 12 Minutes