NULL and **this**

Pointing to Nothing

- What's the default value for a pointer variable?
- Need a distinguished address meaning "not pointing to anything"

The **NULL** Pointer

- **NULL** (defined in **stdlib.h**) can be used to point to nothing
  - Internally, **NULL** is just the memory location **0**
  - The compiler guarantees that nothing will get stored there
- **NULL** is compatible with pointers of every type

Using the **NULL** Pointer

- As a default pointer value:

```
Fraction *pf = NULL;
…
if( pf == NULL ) {
    pf = new Fraction( 22, 7 );
}
```

- As a terminator (soon)

```
for( Node *n = front; n != NULL; n = n->next )
{
    // Do something to data at n
}
```


The Receiver of a Method

```
bool Fraction::isEven()
{
    if( NaN() ) {
        return false;
    }
    return (den == 1) && ((num % 2) == 0);
}
```

- Where do **num**, **den** and **NaN** come from?
  - They are associated with the (implicit) receiver of the method
- But how does this work?
  - How does the method get access to a particular receiver?

The **this** Pointer

- Inside methods, there's a "hidden" extra parameter: **this**
  - **this** always points to the receiver of the method
  - For a method of class **C**, **this** has type pointer-to-**C**
- All implicit references to the receiver are actually referenced via the **this** pointer:

```
bool Fraction::isEven()
{
    if( this->NaN() ) {
        return false;
    }
    return (this->den == 1) && ((this->num % 2) == 0);
}
```

How Classes Work: Declaration

```
class Fraction
{
public:
    Fraction( int n, int d );
    bool NaN();
    bool isEven();
private:
    int num;
    int den;
};
```

```
struct Fraction
{
    int num;
    int den;
};

void Fraction__constructor( Fraction *this, int n, int d );
bool Fraction__isEven( Fraction *this );
bool Fraction__NaN( Fraction *this );
```


How Classes Work: Implementation

```
Fraction::Fraction( int n, int d )
{
    num = n;
    den = d;
}
bool Fraction::isEven()
{
    if( NaN() ) {
        return false;
    }
    return (den == 1) && ((num % 2) == 0);
}
```

```
void Fraction__constructor( Fraction *this, int n )
{
    this->num = n;
    this->den = 1;
}
bool Fraction__isEven( Fraction *this )
{
    if( Fraction__NaN( this ) ) {
        return false;
    }
    return (this->den == 1) && ((this->num % 2) == 0);
}
```


How Classes Work: Client

```
Fraction frac( 5, 1 );

if( frac.isEven() ) {
    cout << "Hello!" << endl;
}
```

```
Fraction frac;
Fraction__constructor( &frac, 5, 1 );

if( Fraction__isEven( &frac ) ) {
    cout << "Hello!" << endl;
}
```


Effective Use of **`this`**

- Use to call other operators with same receiver:

```
bool Fraction::operator !=( Fraction other )
{
    return !( (*this) == other );
}
```

- Use to respect C++ "idiom" (e.g. in assignment operator)

```
Fraction& Fraction::operator =( Fraction other )
{
    num = other.num;
    den = other.den;

    return *this;
}
```


Summary

- **`NULL`** can be used as the value of a pointer that points to nothing
- **`this`** can be used as a way to point to the receiver of a method
  - Classes are implemented by passing **`this`** as a hidden extra parameter in methods