Const methods

- Version 1: Dr. Ofir Pele
- Version 2: Dr. Miri Ben-Nissan
- Version 3: Dr. Erel Segal-Halevi

Const variables – like in c

```
int * const p1 = &i; // a const
// pointer to an un-const variable
```

- `p1++; // c.error`
- `(*p1)++; // ok`

```
const int * p2 = &b; // an un-const
// pointer to a const variable
```

- `p2++; // ok`
- `(*p2)++; // c.error`

```
const int * const p3 = &b; // a const
// pointer to a const variable
```

Const methods *(folder 3)*

```
class A
{ int a;
public:
   void foo1() const;
   void foo2();
};
void A::foo1() const {
    a=5; //error
    cout << a; // OK
}
void A::foo2()
{
}
```

```
int main()
{
   A a;
   const A ca;
   a.foo1();
   a.foo2();
   ca.foo1();
   ca.foo2();
    // comp. error
}
```

Const methods

```
class A
{
public:
    void foo() const;
    void foo();
};
const int A::foo() const
{
    cout << "const foo\n";
}
int& A::foo()
{
    cout << "foo\n";
}
```

```
int main()
{
    A a;
    const A ca;
    a.foo () = 5;
    ca.foo();
}
```

```
// output
foo
const foo
```

Why?

Overload resolution, again:

```
A::foo(A* this)

A::foo(const A* this)
```

mutable

- `mutable` means that a variable can be changed by a const function (even if the object is const)
- QUESTION: When would you use this?

mutable: example #1

```
class X
{
public:
  ...
  X() : _fooAccessCount(0) {}

  bool foo() const
   {
      ++_fooAccessCount;
      ...
   }
   unsigned int fooAccessCount() { return _fooAccessCount; }

private:
   mutable unsigned int _fooAccessCount;
};
```

mutable: example #2

```
class Shape
{
public:
  ...
  void set...(...) { _areaNeedUpdate= true; ... }
  double area() const
   {
      if (_areaNeedUpdate) {
         _area = ...
         _areaNeedUpdate= false;
      }
      return _area;
   }
private:
   mutable bool _areaNeedUpdate= true;
   mutable double _area;
};
```