# Introduction to C++

David Lawrence, JLab

July 17, 2008

# What C++ is good for

- Writing programs that run natively on the local processor (fast)
- Making very large projects modular

# Where not to use C++

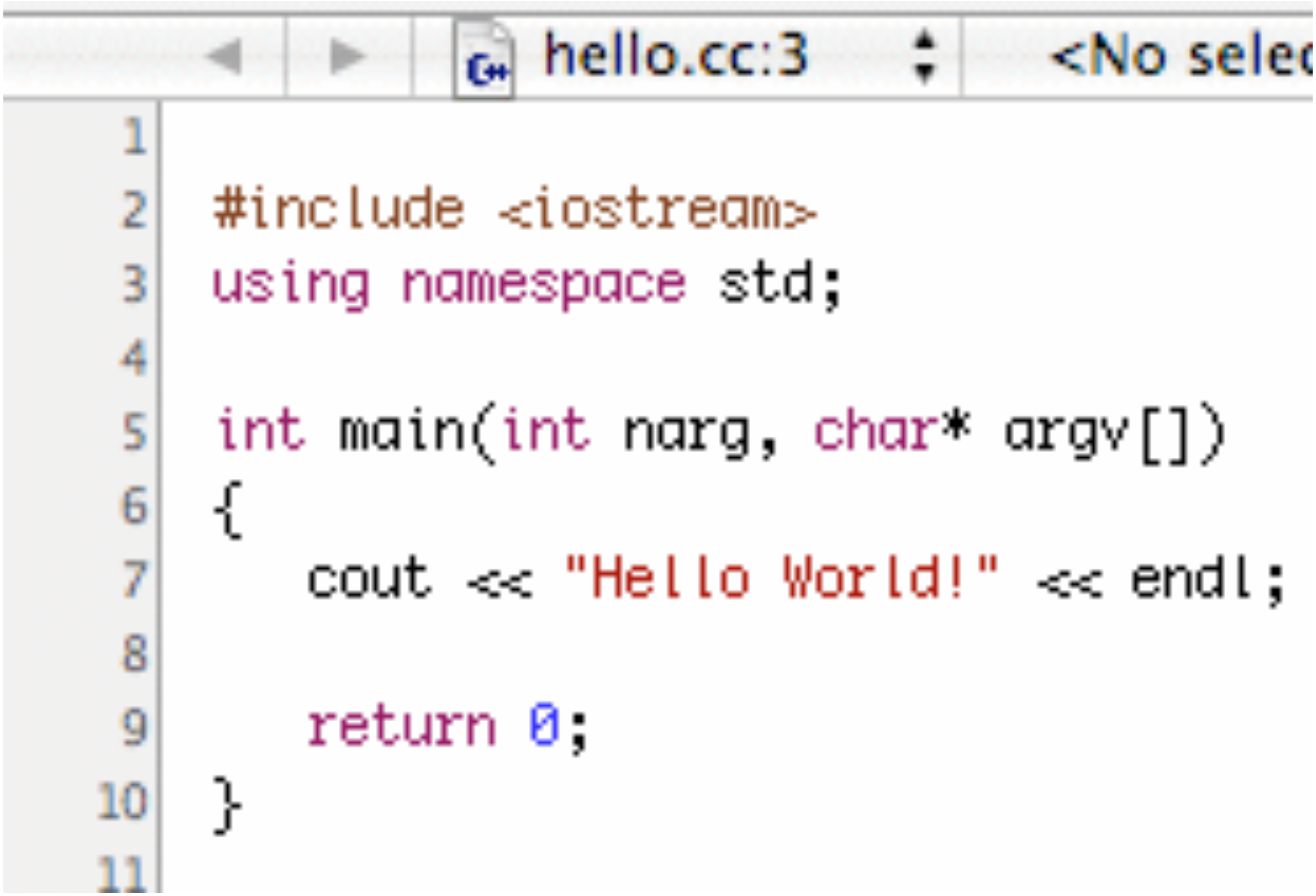
- Short programs focused on parsing text (consider using a script)

- Programs with wide distribution across many platforms but without severe efficiency requirements (consider Java)

# C vs. C++

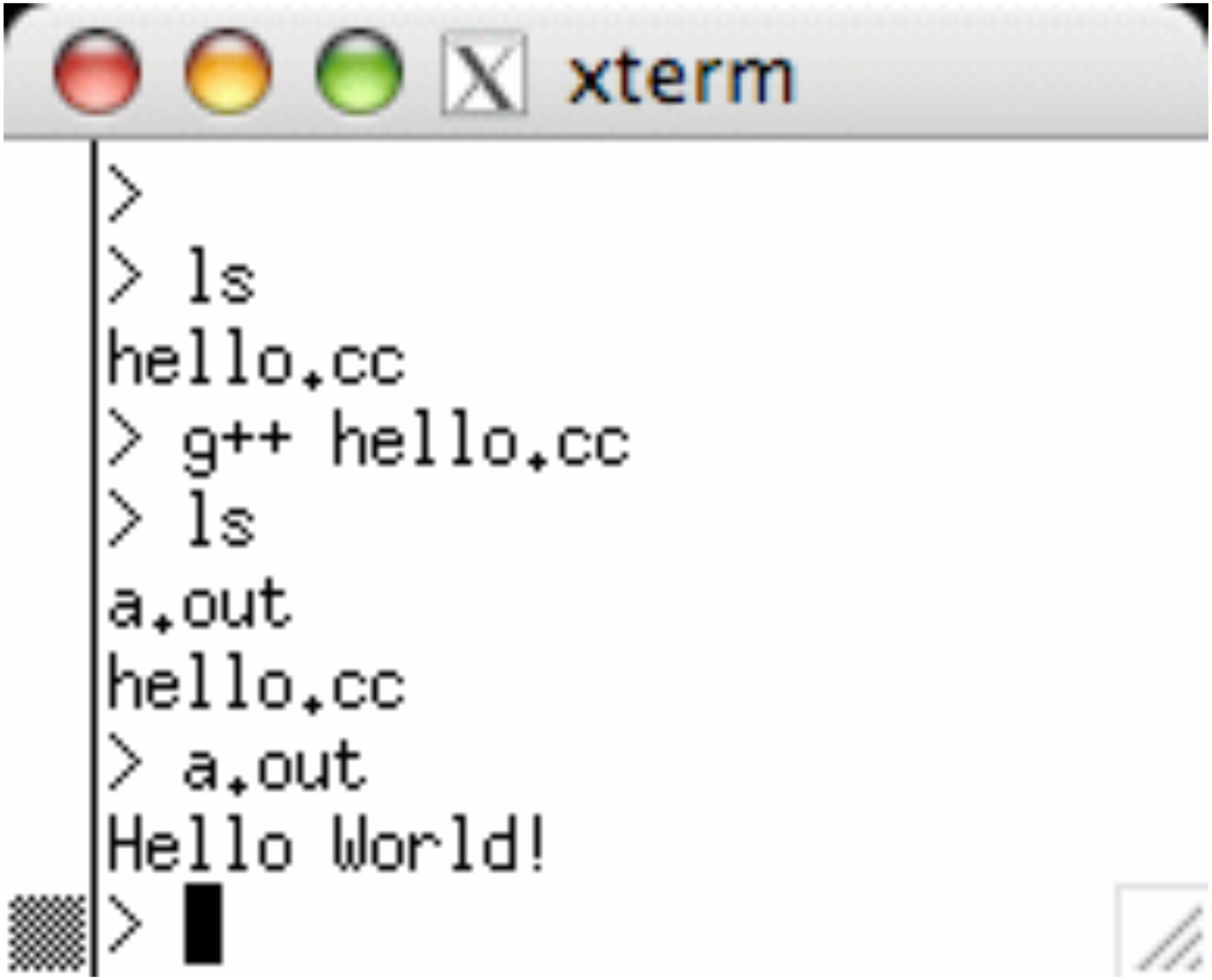
- A C++ compiler will compile and run C code, but not vice versa
- C++ can be used to develop *object oriented* code while C can’t
- *C++ is “+1” better than C*

# “Hello World!”



```

#include <iostream>
using namespace std;

int main(int narg, char* argv[])
{
    cout << "Hello World!" << endl;

    return 0;
}

```

# Compiling, Linking, and Running a program



```
>
> ls
hello.cc
> g++ hello.cc
> ls
a.out
hello.cc
> a.out
Hello World!
> █
```

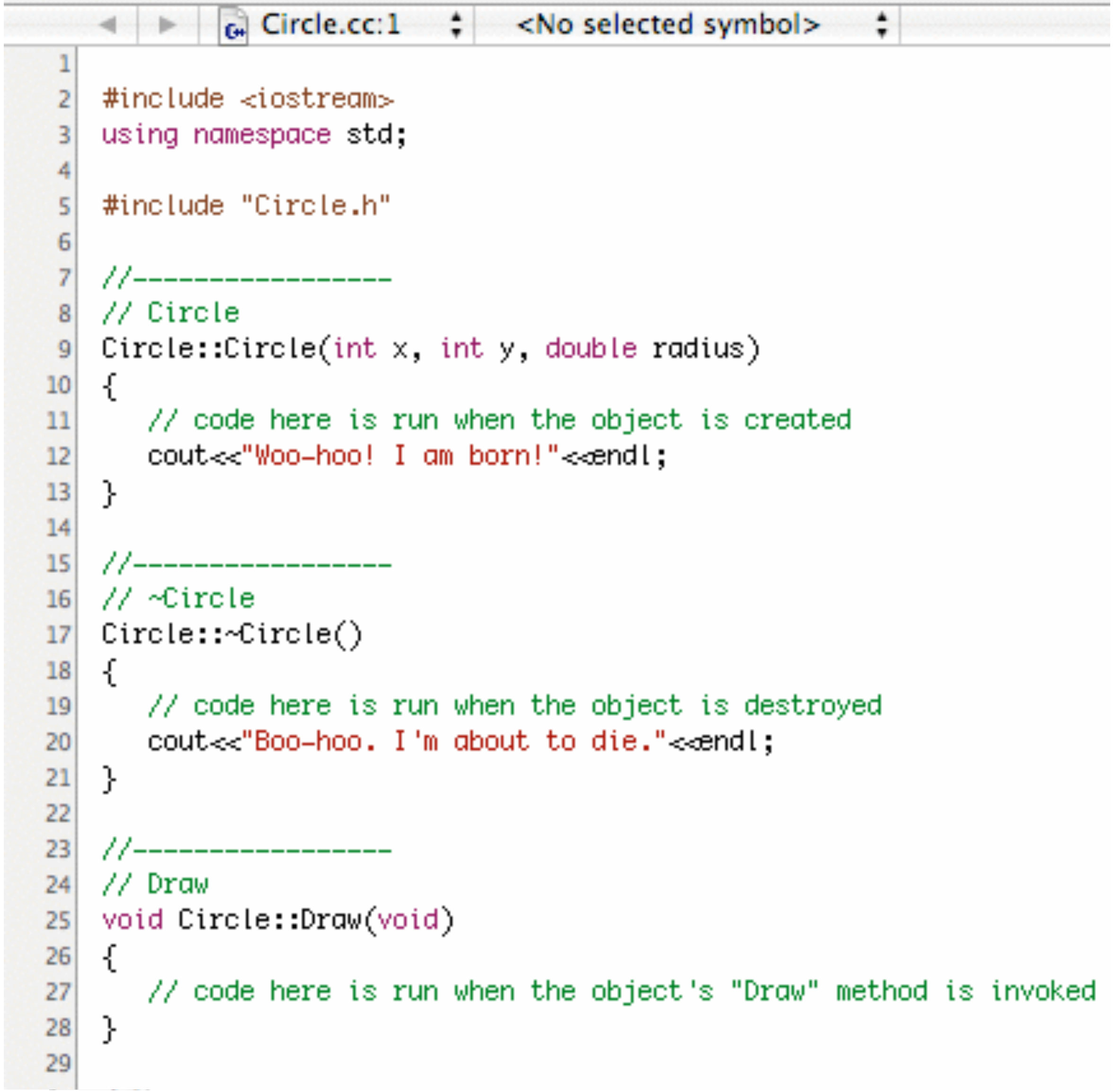
# Object Oriented Programming

**class**: The definition of what an object will contain when it is created.

**object**: A real instance of a class.

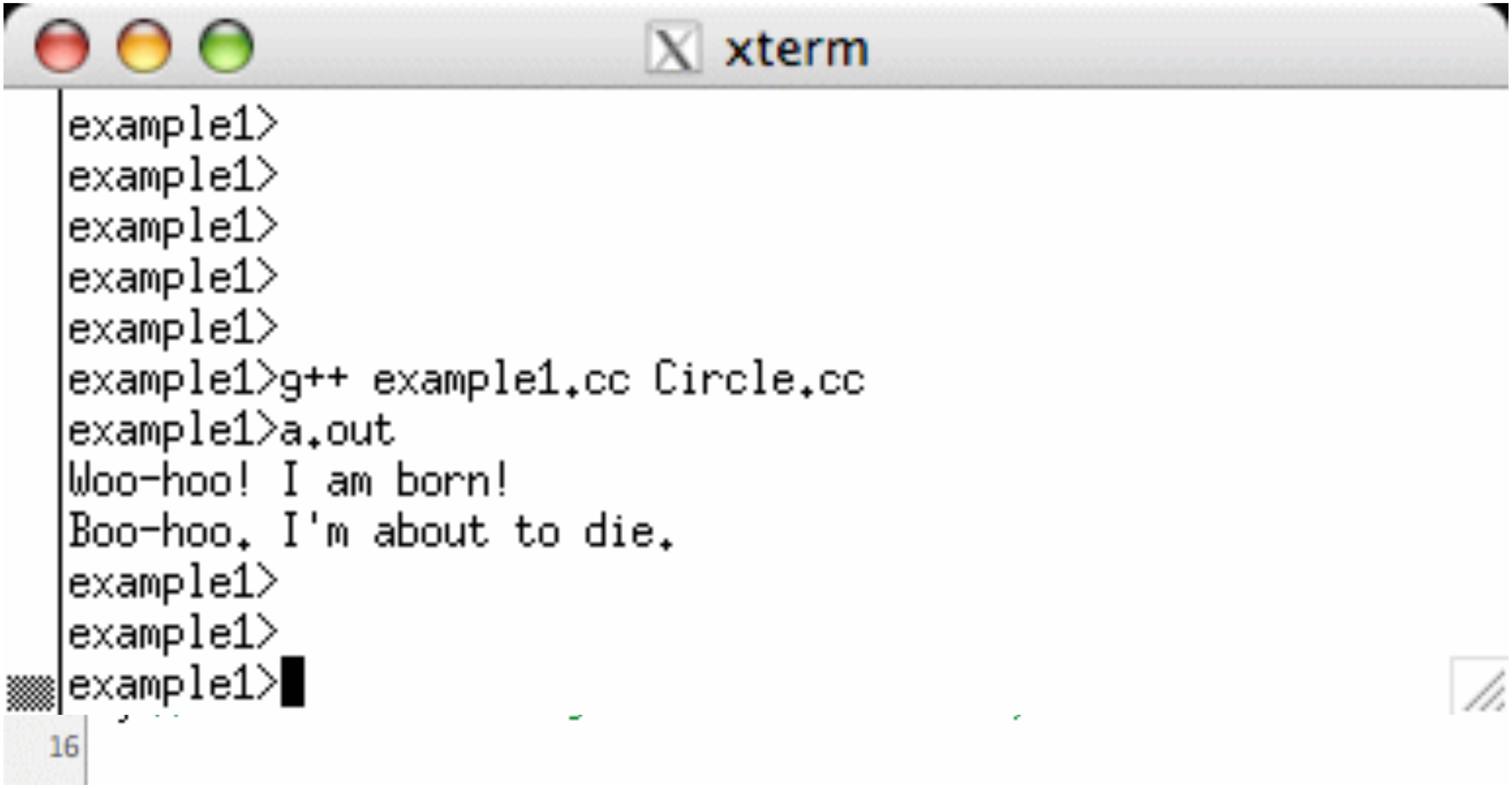
**struct**: C-style data structure that does not contain methods

# Defining a Class



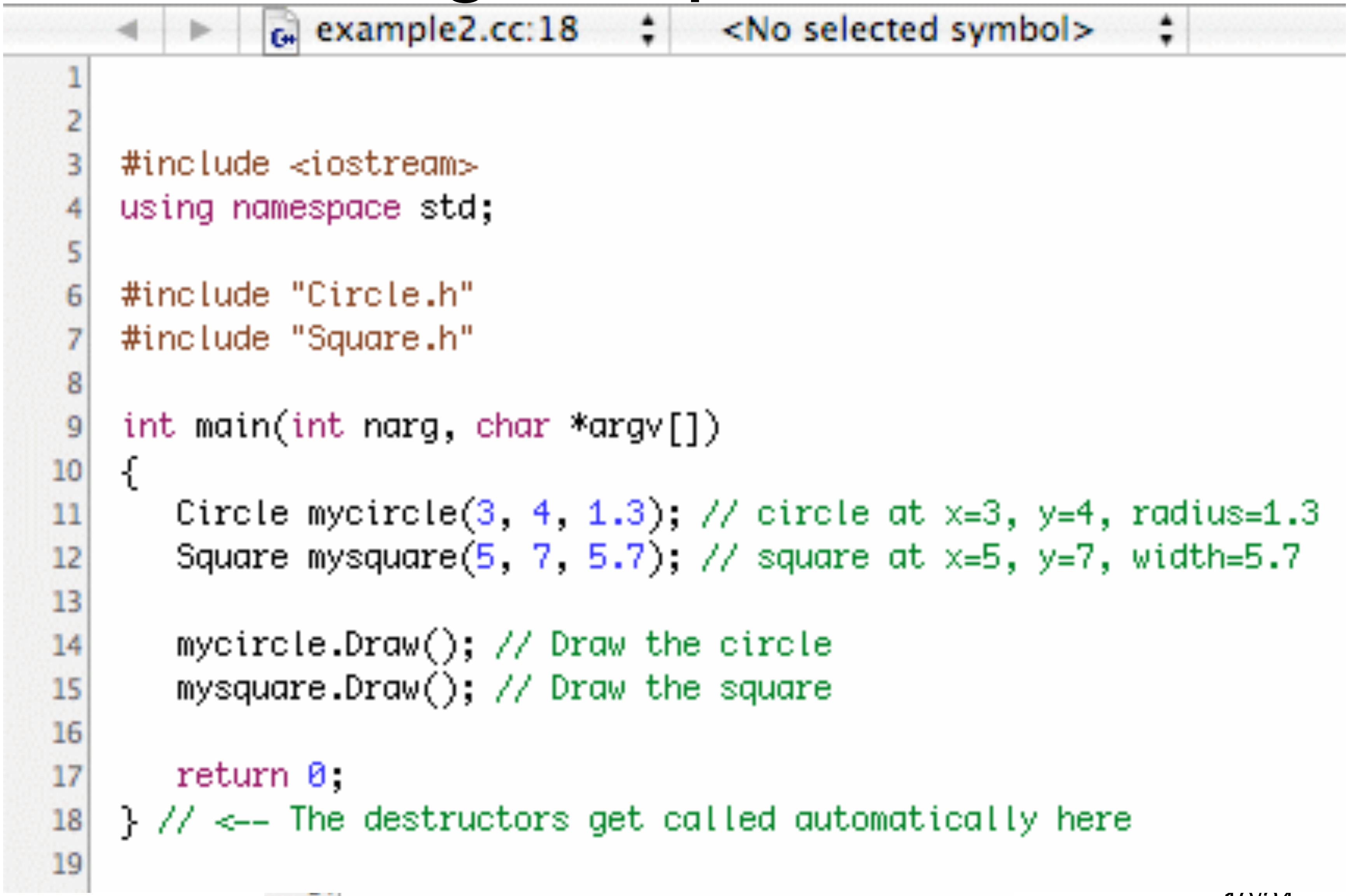
```

#include <iostream>
using namespace std;

#include "Circle.h"

//-----------------
// Circle
Circle::Circle(int x, int y, double radius)
{
   // code here is run when the object is created
   cout<<"Woo-hoo! I am born!"<<endl;
}

//-----------------
// ~Circle
Circle::~Circle()
{
   // code here is run when the object is destroyed
   cout<<"Boo-hoo. I'm about to die."<<endl;
}

//-----------------
// Draw
void Circle::Draw(void)
{
   // code here is run when the object's "Draw" method is invoked
}

```

# Using the Circle Class



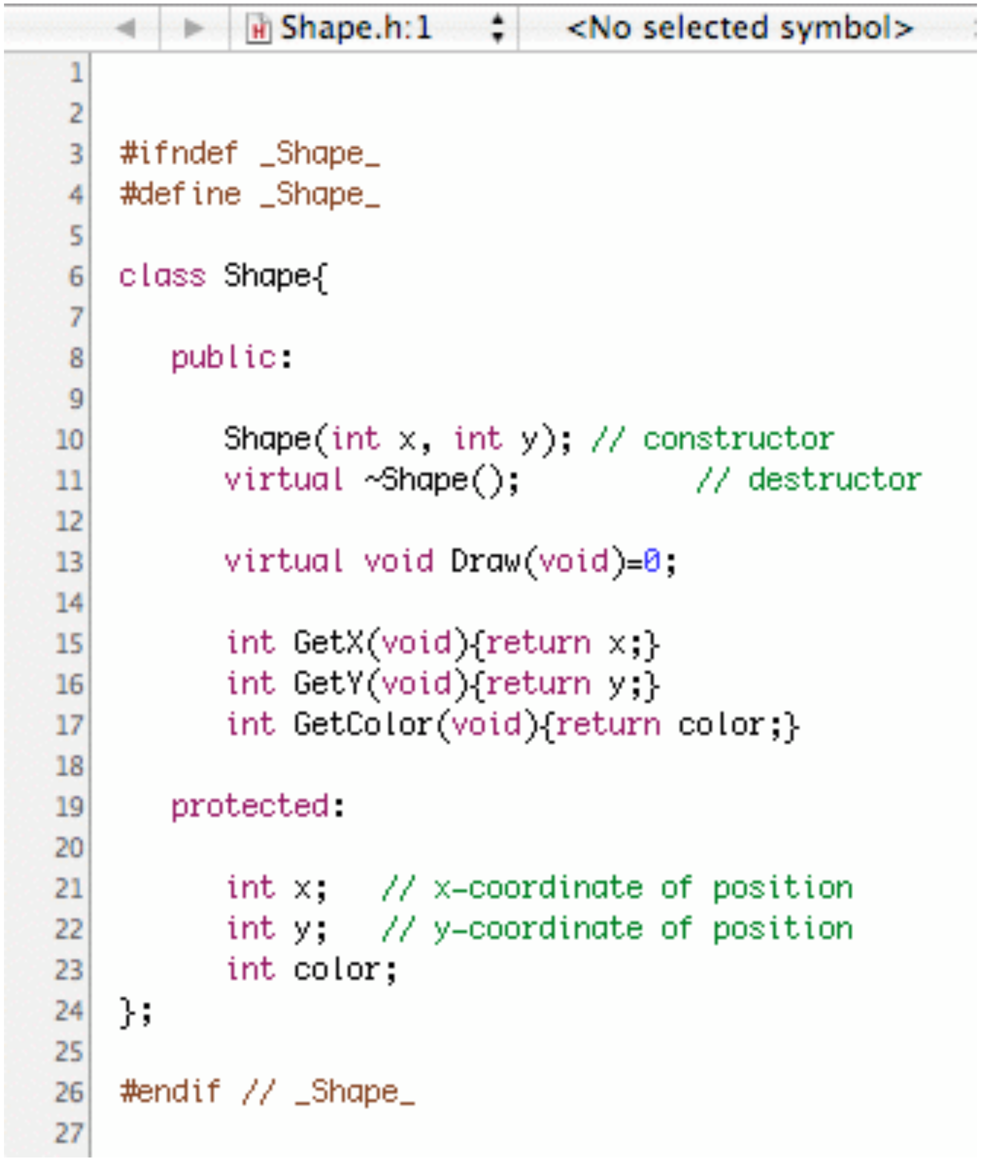
```
example1>
example1>
example1>
example1>
example1>
example1>g++ example1.cc Circle.cc
example1>a.out
Woo-hoo! I am born!
Boo-hoo. I'm about to die.
example1>
example1>
example1>
```

# Adding a Square Class



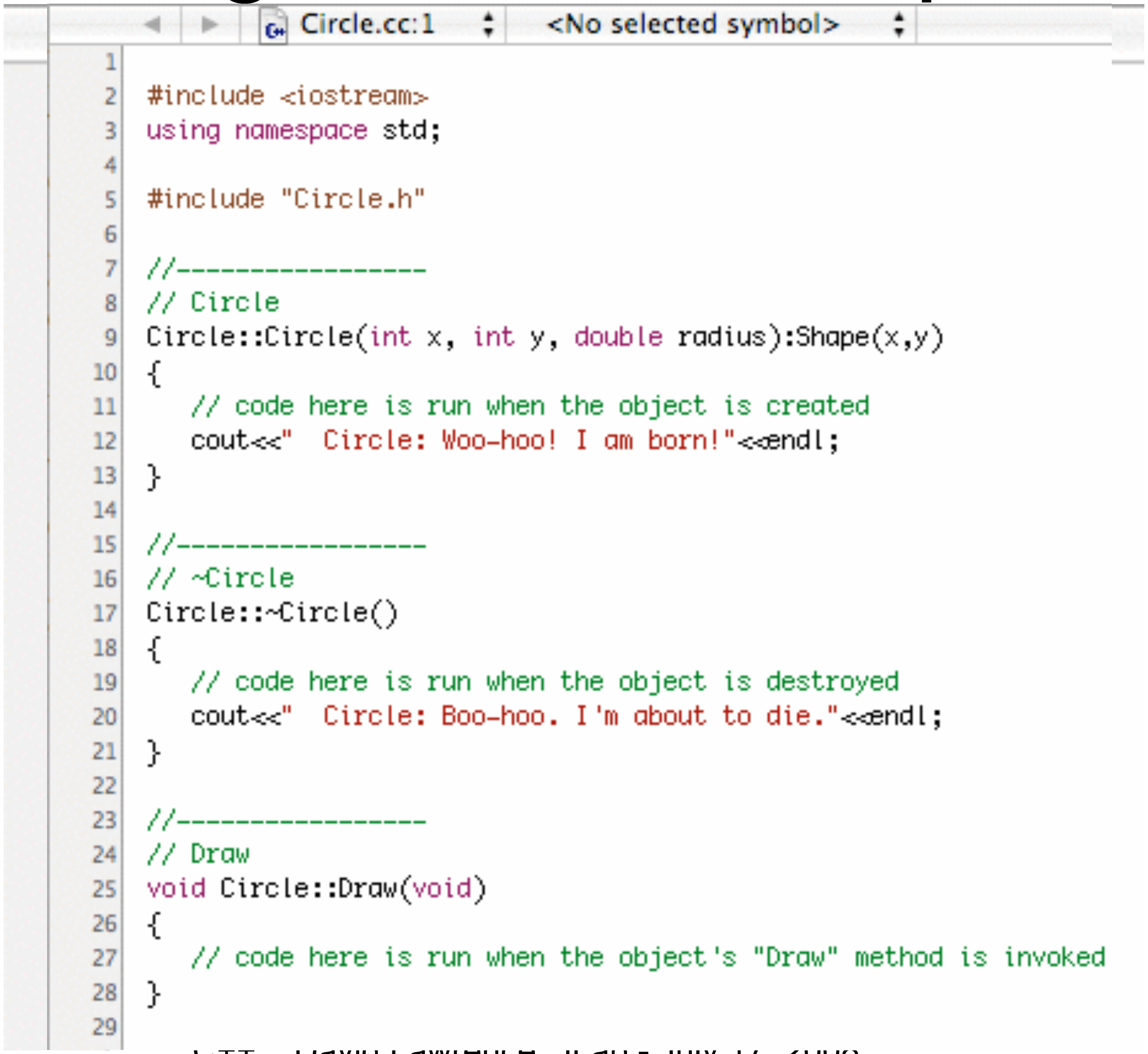
```cpp


#include <iostream>
using namespace std;

#include "Circle.h"
#include "Square.h"

int main(int narg, char *argv[])
{
   Circle mycircle(3, 4, 1.3); // circle at x=3, y=4, radius=1.3
   Square mysquare(5, 7, 5.7); // square at x=5, y=7, width=5.7

   mycircle.Draw(); // Draw the circle
   mysquare.Draw(); // Draw the square

   return 0;
} // <-- The destructors get called automatically here

```

# Defining the Shape Class



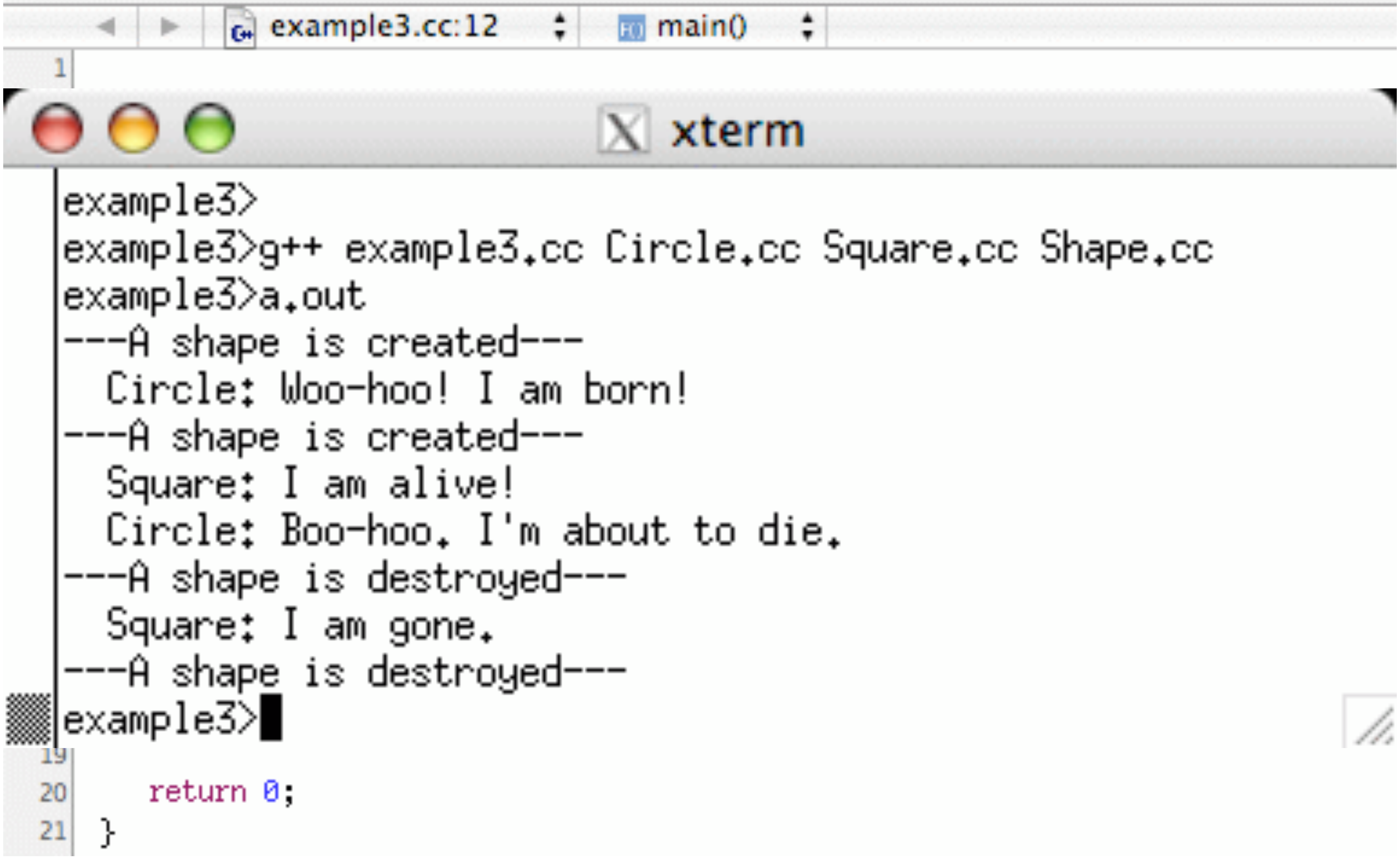
```cpp


#ifndef _Shape_
#define _Shape_

class Shape{

   public:

      Shape(int x, int y); // constructor
      virtual ~Shape();          // destructor

      virtual void Draw(void)=0;

      int GetX(void){return x;}
      int GetY(void){return y;}
      int GetColor(void){return color;}

   protected:

      int x;   // x-coordinate of position
      int y;   // y-coordinate of position
      int color;
};

#endif // _Shape_

```

# Inheriting from the Shape Class



```

#include <iostream>
using namespace std;

#include "Circle.h"

//-----------------
// Circle
Circle::Circle(int x, int y, double radius):Shape(x,y)
{
   // code here is run when the object is created
   cout<<"  Circle: Woo-hoo! I am born!"<<endl;
}

//-----------------
// ~Circle
Circle::~Circle()
{
   // code here is run when the object is destroyed
   cout<<"  Circle: Boo-hoo. I'm about to die."<<endl;
}

//-----------------
// Draw
void Circle::Draw(void)
{
   // code here is run when the object's "Draw" method is invoked
}

```

# Polymorphism



```
example3>
example3>g++ example3.cc Circle.cc Square.cc Shape.cc
example3>a.out
---A shape is created---
  Circle: Woo-hoo! I am born!
---A shape is created---
  Square: I am alive!
  Circle: Boo-hoo. I'm about to die.
---A shape is destroyed---
  Square: I am gone.
---A shape is destroyed---
example3>
```

```
    return 0;
}
```

# Inheritance and Polymorphism Defined

**Inheritance**: C++ classes can be *derived* from one another. The derived class *inherits* the base class's attributes.

**Polymorphism**: The ability of a derived object to appear as though it is one of its base classes.

# Public/Protected/Private

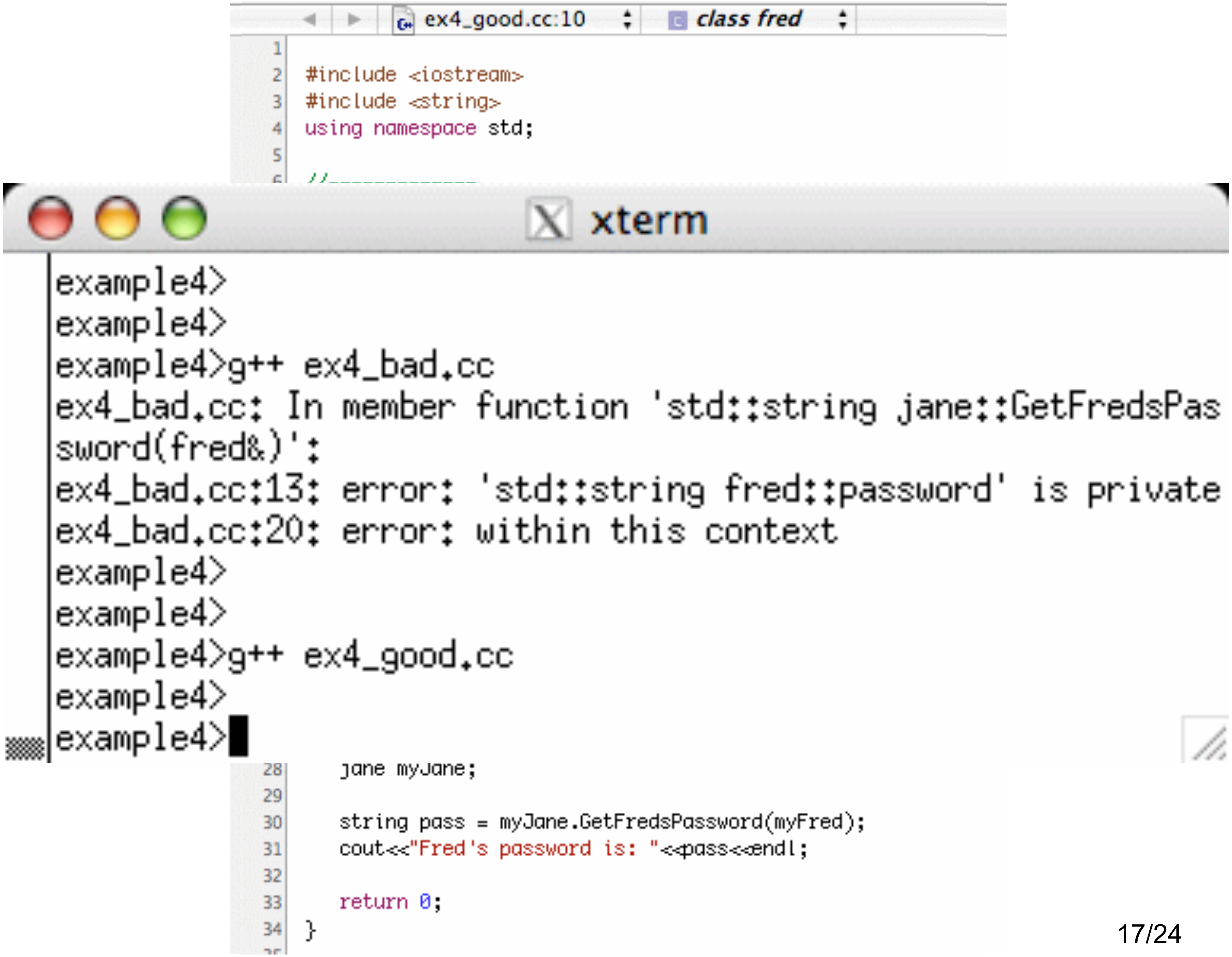
**Access restrictions are NOT intended to prevent folks with devious intentions from accessing your class.**

**They are to help prevent you (and others) from shooting yourself in the foot!**

members accessible only by this class

# “Friend”s

A class may share its private members with other classes by declaring them as a “friend”.



```
#include <iostream>
#include <string>
using namespace std;

//------------
```

```
example4>
example4>
example4>g++ ex4_bad.cc
ex4_bad.cc: In member function 'std::string jane::GetFredsPassword(fred&)':
ex4_bad.cc:13: error: 'std::string fred::password' is private
ex4_bad.cc:20: error: within this context
example4>
example4>
example4>g++ ex4_good.cc
example4>
example4>
```

```
    jane myJane;

    string pass = myJane.GetFredsPassword(myFred);
    cout<<"Fred's password is: "<<pass<<endl;

    return 0;
}
```

# Primitive types

- char (unsigned char)
- int (unsigned int)
- long (unsigned long)
- float
- double
- bool
- void

**string**

The ANSI string class is not a primitive, but should be the basis of most code dealing with strings

# Qualifiers

- **const**

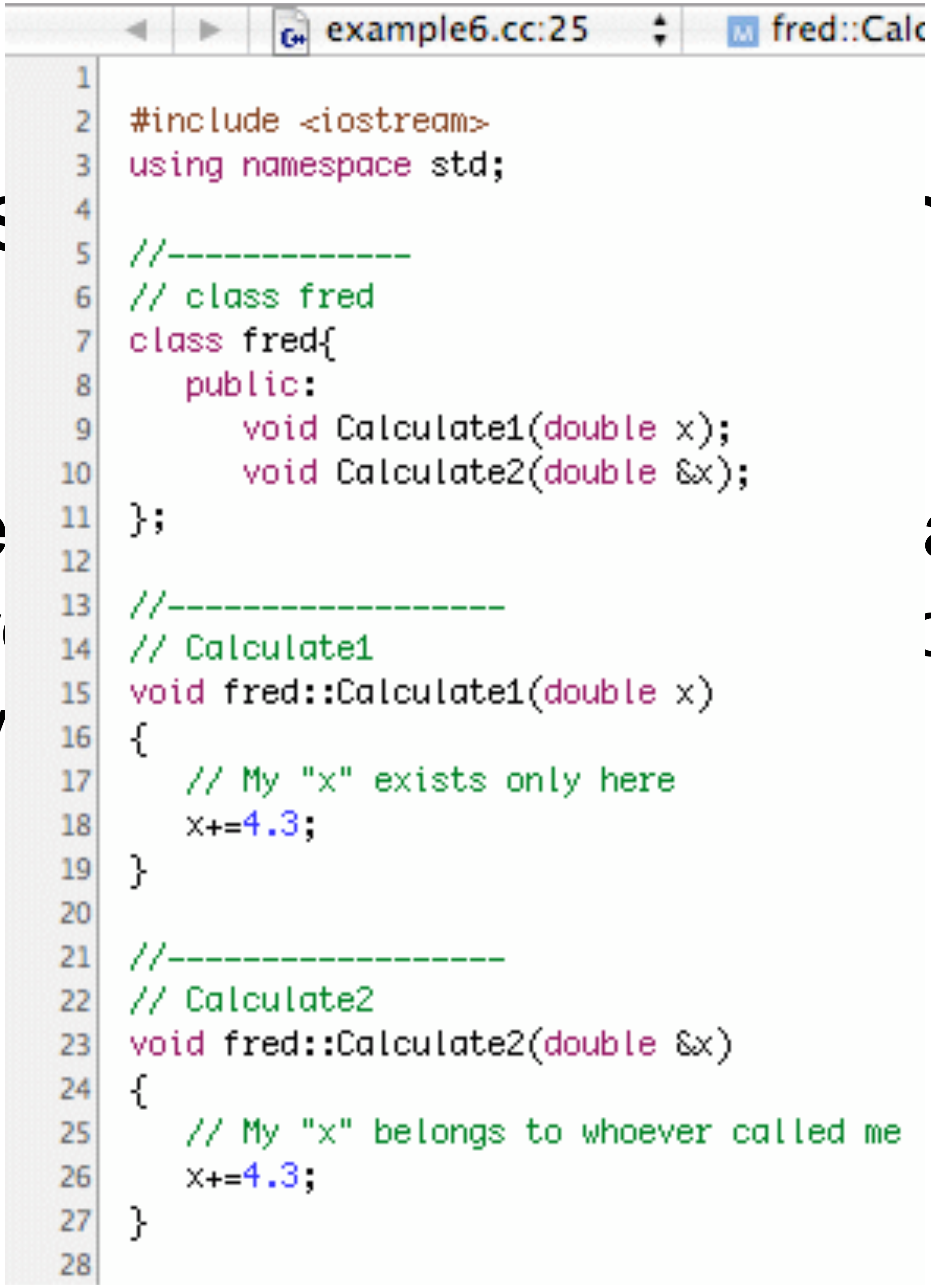
  defining a variable as const indicates that either it can’t be changed or what it points to can’t be changed

- **static**

  defining something as static means it is not deleted when it goes out of scope

# Pas ' value

- Argu passed eithe assing by refer change the v



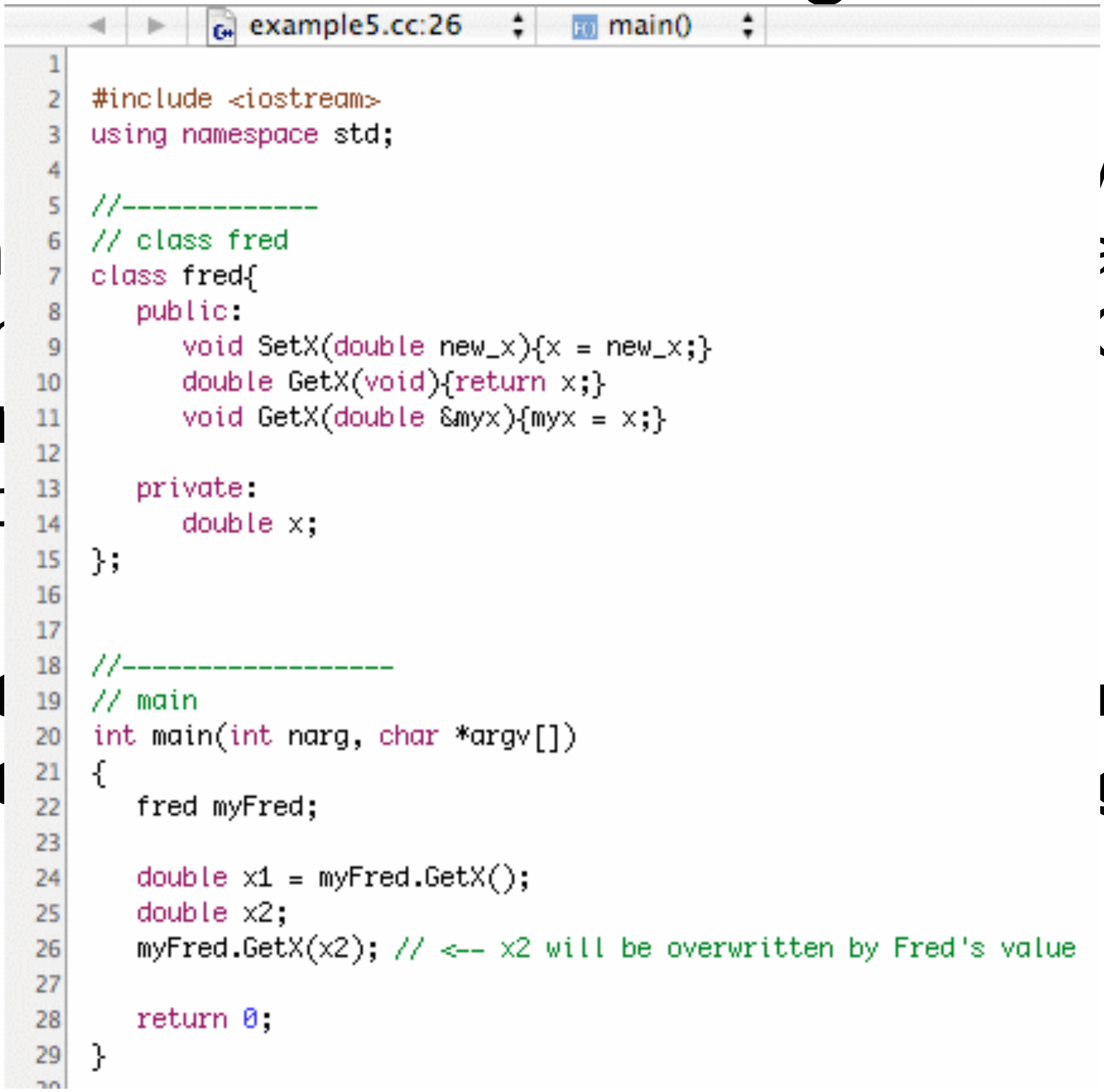
```
#include <iostream>
using namespace std;

//-------------
// class fred
class fred{
   public:
      void Calculate1(double x);
      void Calculate2(double &x);
};

//-----------------
// Calculate1
void fred::Calculate1(double x)
{
   // My "x" exists only here
   x+=4.3;
}

//-----------------
// Calculate2
void fred::Calculate2(double &x)
{
   // My "x" belongs to whoever called me
   x+=4.3;
}

```

# Overloading

- C++ / a
  meth ame of
  the m OT the
  retur are
  comp

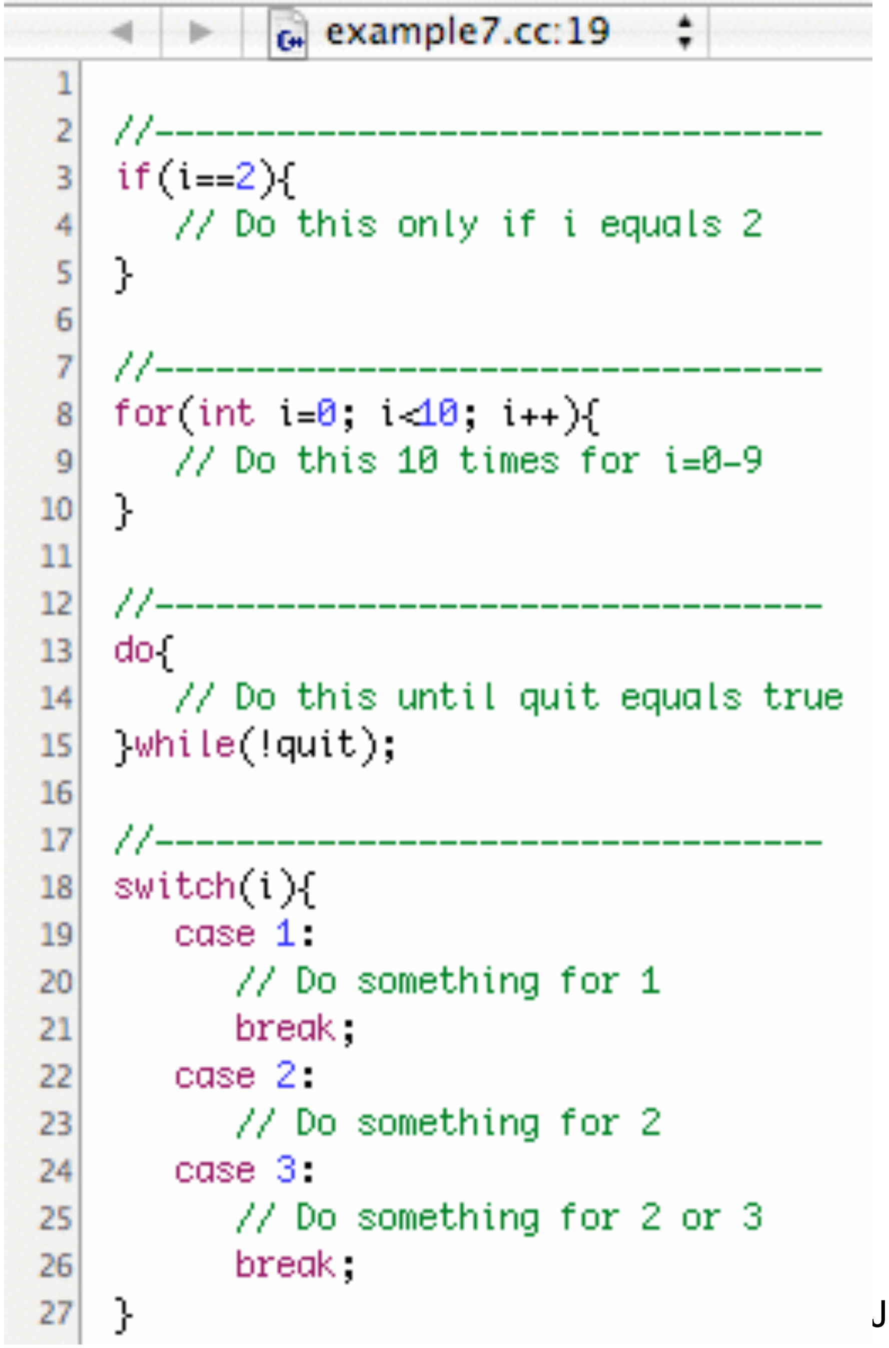
**int Fi npars);**

**int Fi g npars);**



```cpp

#include <iostream>
using namespace std;

//-------------
// class fred
class fred{
    public:
        void SetX(double new_x){x = new_x;}
        double GetX(void){return x;}
        void GetX(double &myx){myx = x;}

    private:
        double x;
};


//-----------------
// main
int main(int narg, char *argv[])
{
    fred myFred;

    double x1 = myFred.GetX();
    double x2;
    myFred.GetX(x2); // <-- x2 will be overwritten by Fred's value

    return 0;
}
```

# Flow Control

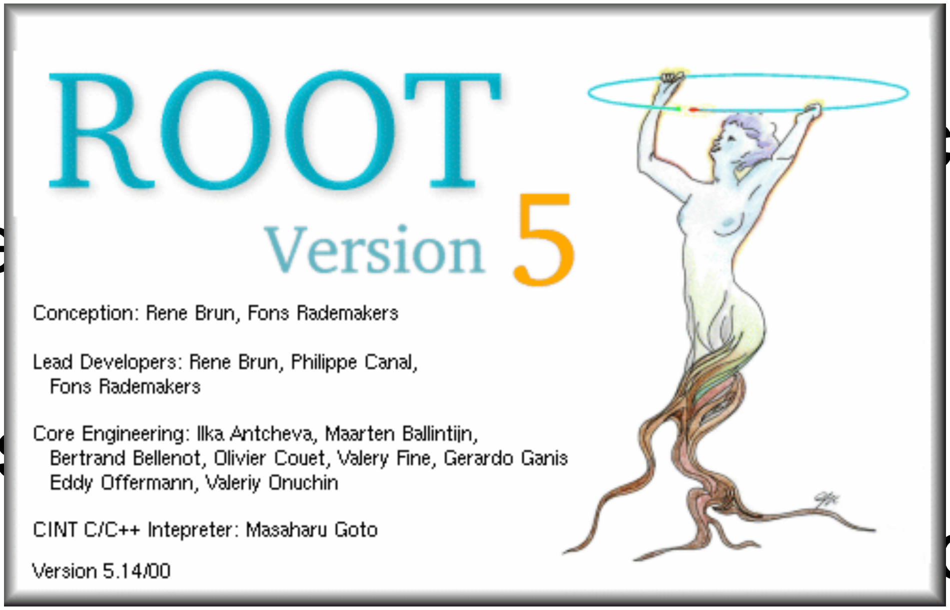


```cpp

//--------------------------------
if(i==2){
   // Do this only if i equals 2
}

//--------------------------------
for(int i=0; i<10; i++){
   // Do this 10 times for i=0-9
}

//--------------------------------
do{
   // Do this until quit equals true
}while(!quit);

//--------------------------------
switch(i){
   case 1:
      // Do something for 1
      break;
   case 2:
      // Do something for 2
   case 3:
      // Do something for 2 or 3
      break;
}
```

# ROOT

- ROOT very
  widely lear and
  particle

  It is ba a built-
  in C++ o write
  and execute C++ code interactively



Conception: Rene Brun, Fons Rademakers

Lead Developers: Rene Brun, Philippe Canal,
Fons Rademakers

Core Engineering: Ilka Antcheva, Maarten Ballintijn,
Bertrand Bellenot, Olivier Couet, Valery Fine, Gerardo Ganis
Eddy Offermann, Valeriy Onuchin

CINT C/C++ Intepreter: Masaharu Goto

Version 5.14/00

# Summary

- C++ is an object-oriented language
- There are many features to help you write robust, modular code (but you have to use them!)
- Come see Elliott Wolin’s talk on C++ on Friday (week from tomorrow)