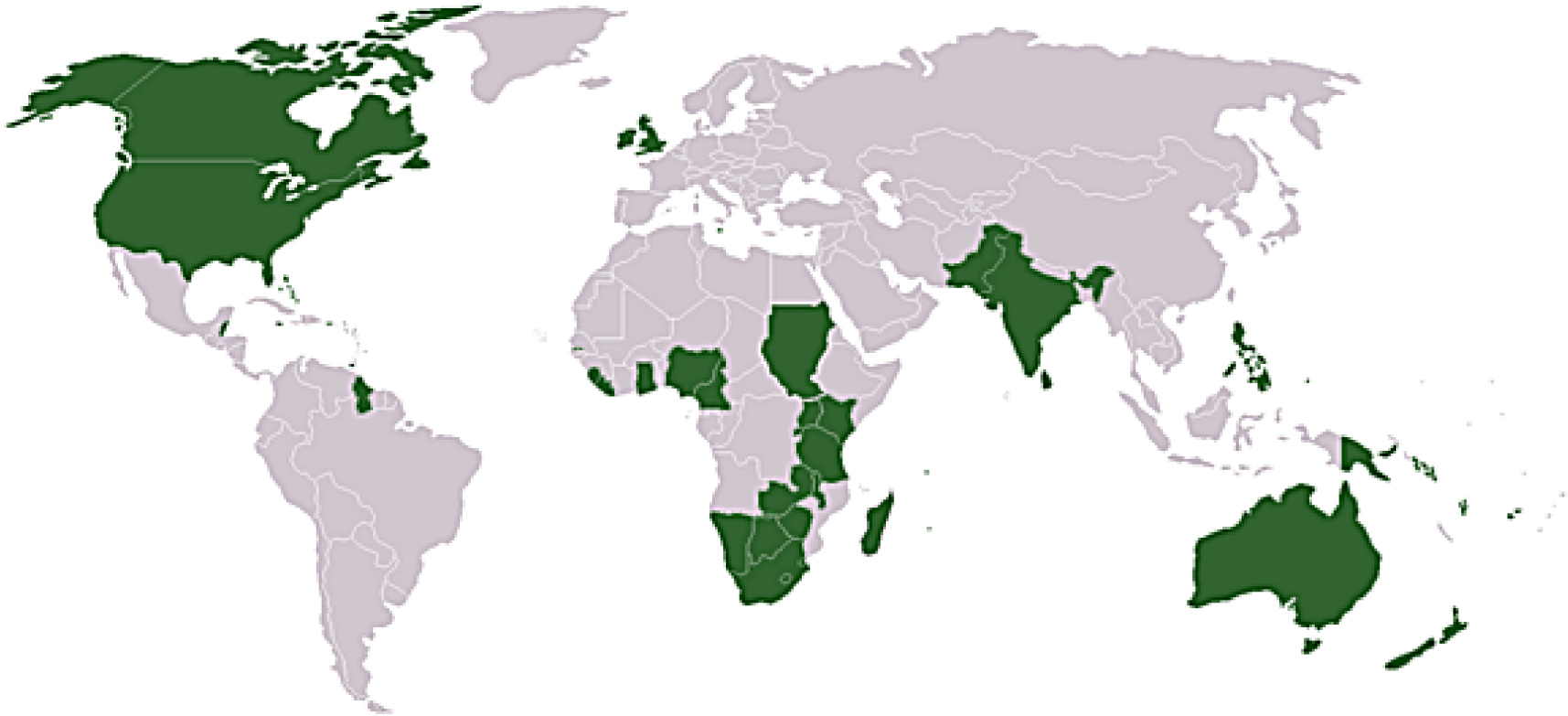
Introduction to C++

Adapted from Jamie Andrews, Winter 2010

"The easiest way is to view C++ not as a single language but as a federation of related languages"

Scott Meyers - Effective C++



Goals:

Give you a basic understanding of C++ structure, to enable you to learn the language specifics on your own.

Reminder of important concepts from C, namely pointers.

Give you a heads up on issues you may encounter when getting accustomed to C++, that are not common in other languages.

Today

Introduction to C++

C++ Class structure

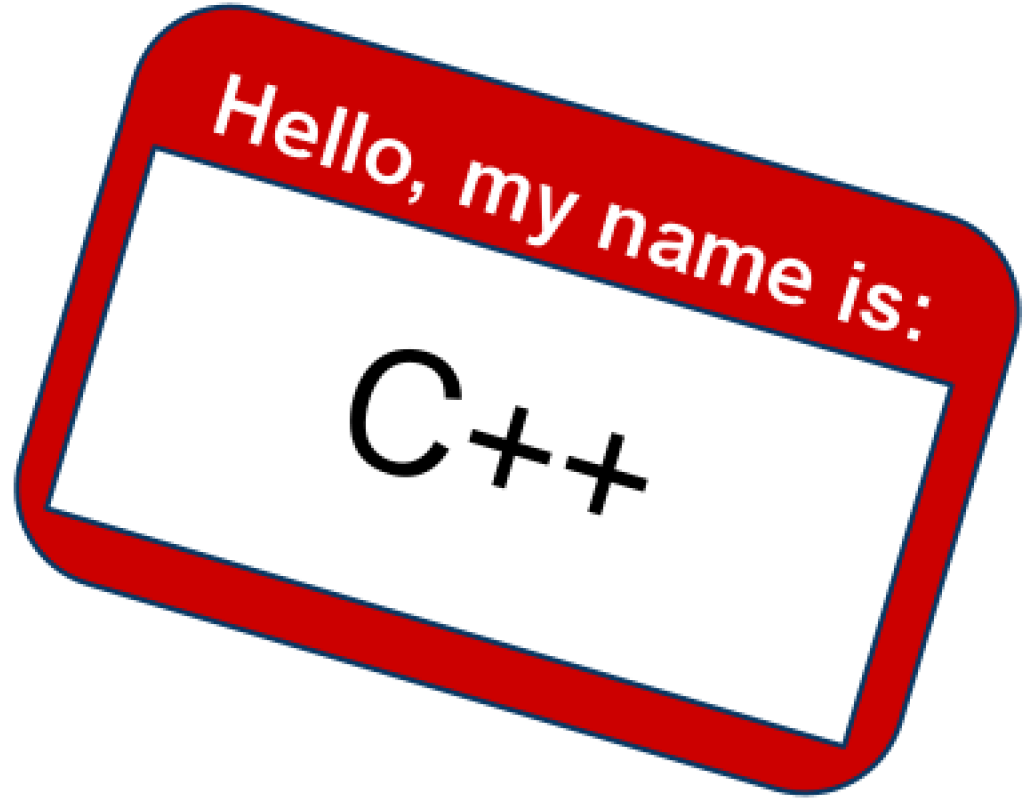
Pointers, Briefly

Separate Compilation

Namespaces

Compiler Generated Code

Introduction to C++



"Middle-level" language
both:
- Procedural (low level)
- Object Oriented (high level)

Designed to give the programmer choice
- Enough rope

C++ is like C

- Much of the basic syntax is the same
- Compiles to machine code
- Manual memory management
- Pointers
- Statically typed
- Relies on libraries

```
#include  <iostream>

int main()
{
    std::cout<< "I know how to add!" << std::endl;
    int a, b, sum;
    std::cin >> a;
    std::cin >> b;
    sum = a + b;
    std::cout<< a <<"+"<< b <<"="<< sum << std::endl;
}
```

Object Oriented C++

A Class is described by two things:

- Declaration ("ClassName.h")
  - Name for the class
  - Declares members and their visibility
    - member function prototypes (headers)
    - data members

- Definitions ("ClassName.cpp")
  - Implementation of functions

Java - Example Class definition for Circle, "Circle.java"

```
public class Circle {

    private double radius = 0;
    const double PI = 3.14159;

    Circle(double radius){
        this.radius = radius;
    }

    public double getArea() {
        return PI * radius * radius;
    }
}
```

C++: Example Class Declaration for Circle, "Circle.h"

```
class Circle {

    public:
        Circle(double radius);
        double area();

    protected:

    private:
        int radius = 0;
        const double PI = 3.14159;

};      //SEMICOLON!!!!
```

**Note: We will add to this later*

C++ Example Function Definitions File "Circle.cpp"

```
#include "Circle.h"

Circle::Circle(double radius)
{
    this->radius = radius;
}

double Circle::area()
{
    return PI * radius * radius;
}
```

Object Oriented C++

A Class is described by two things:

- Declaration ("ClassName.h")
  - Name for the class
  - Declares members and their visibility
    - member function prototypes (headers)
    - data members

- Definitions ("ClassName.cpp")
  - Implementation of functions

But

- Not required for object oriented programming

- Standard convention to have header file and implementation separated

- Can mix (perhaps inadvertently) procedural and object oriented elements

Object Orientation is a Feature of C++

Only by following conventions is true object oriented programming achieved

Today

Introduction to C++

C++ Class structure

Pointers, Briefly

Separate Compilation

Namespaces

Compiler Generated Code

Pointers

Pointers have two pieces of information
- A memory address
- The type (or void) of what is at that address

```
int a = 5;
int * b = NULL;
b = &a;
*b = 10;
std::cout << a << std::endl;
```

`*` translates to "pointer" `int * b`

`(int*) b` - b holds an int pointer
`int (*b)` - pointer b holds an int

`*` translates to "pointer" **`int * b`**

`(int*) b` - b holds an int pointer
`int (*b)` - pointer b holds an int

`&` translates to "reference" **`& a`**

- gets the address of the variable a

`*` translates to "pointer" **`int * b`**

`(int*) b` - b holds an int pointer
`int (*b)` - pointer b holds an int

`&` translates to "reference" **`& a`**

- gets the address of the variable a

`b = &a;`
- pointer can only hold that address if the types match

Pointers to Objects

```
Circle * myCircle = new Circle(5);

double myArea;

myArea = *myCircle.area();
```

Pointers to Objects

```
Circle * myCircle = new Circle(5);

double myArea;

//myArea = *myCircle.area();     //NOPE
myArea = (*myCircle.area());
```

Pointers to Objects

```
Circle * myCircle = new Circle(5);

double myArea;

//myArea = *myCircle.area();     //NOPE
//myArea = (*myCircle.area());   //NOPE
```

Pointers to Objects

```
Circle * myCircle = new Circle(5);

double myArea;

//myArea = *myCircle.area();     //NOPE
//myArea = (*myCircle.area());   //NOPE

myArea = (*myCircle).area();
```

Pointers to Objects

```
Circle * myCircle = new Circle(5);

double myArea;

//myArea = *myCircle.area();     //NOPE
//myArea = (*myCircle.area());   //NOPE

//myArea = (*myCircle).area();   //OK
myArea = myCircle->area();       //Best :)
```

Today

Introduction to C++

C++ Class structure

Pointers, Briefly

Separate Compilation

Namespaces

Compiler Generated Code

Separate Compilation

A Few Facts:

- C++ files are compiled one at a time
    - vs Java, where all classes used compile together

- Classes must be declared before they are used

- Class declarations can be separated from implementations
    - .h file vs .cpp file

#include "declaration.h"

- Compiler directive

- Treat the contents of the file as if it were included in the source code where the #include statement is.

Recall Circle.h

```
class Circle {

    public:
        Circle(double radius);
        double area();

    private:
        int radius = 0;
        const double PI = 3.14159;

};      //SEMICOLON!!!!
```

and Circle.cpp

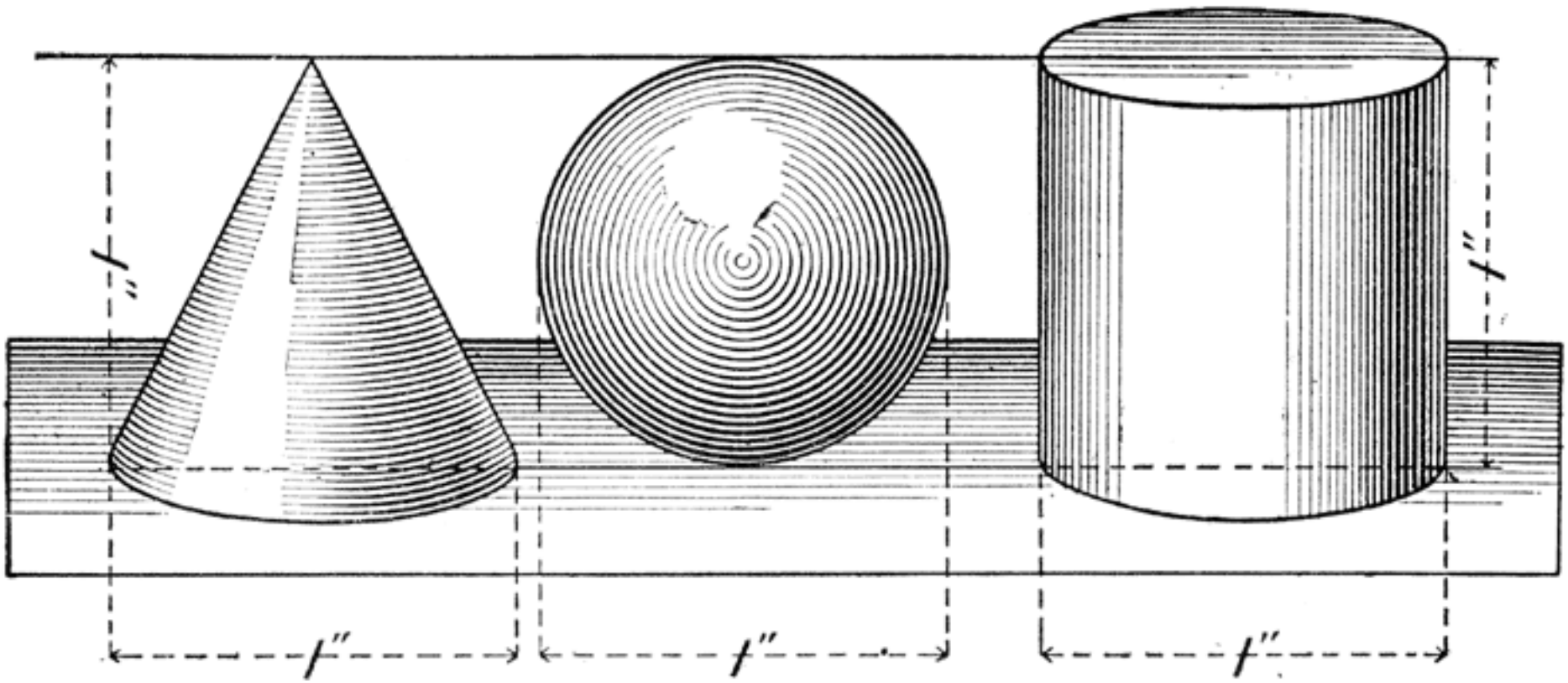
```
#include "Circle.h"

Circle::Circle(double radius)
{
    this->radius = radius;
}

double Circle::area()
{
    return PI * radius * radius;
}
```

```
class Circle {

    public:
        Circle(double radius);
        double area();

    private:
        int radius = 0;
        const double PI = 3.14159;

};      //SEMICOLON!!!!

Circle::Circle(double radius)
{
    this->radius = radius;
}

double Circle::area()
{
    return PI * radius * radius;
}
```

#include "Circle.h"

Why have separate files?

- In large projects, many files may use the same class

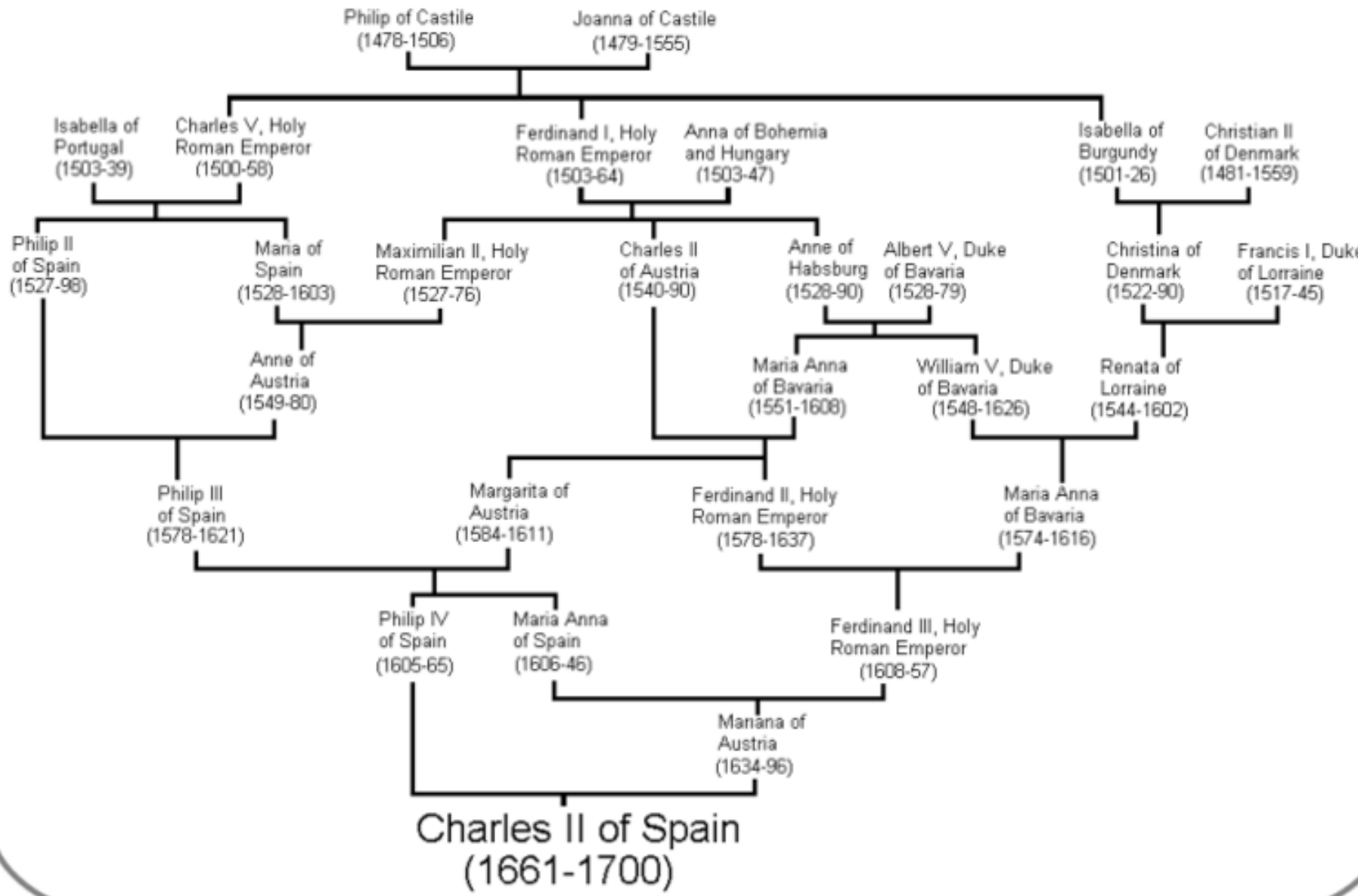


Includes can accumulate

- If a file you #include also #includes files...you get them all

- Only include files you need,
    - speed up compile time
    - more readable

The Ancestry of King Charles II of Spain (1661-1700)



The Include Guard

- used to ensure a file does not unintentionally include more than one declaration for a class

```
#ifndef CIRCLE_H
#define CIRCLE_H

class Circle {

};      //SEMICOLON!!!!

#endif
```

Circular Declarations

the #include guard does not protect you from this:

```
#ifndef BAR_H
#define BAR_H

class foo{
...
    private
        Bar barObj;
};

#endif
```

```
#ifndef FOO_H
#define FOO_H

class Bar{
...
    private
        Foo fooObj;
};

#endif
```

Today

Introduction to C++

C++ Class structure

Pointers, Briefly

Separate Compilation

Namespaces

```
class Circle {

    public:
        Circle(double radius);
        double area();

    private:
        int radius = 0;
        const double PI = 3.14159;

};      //SEMICOLON!!!!

Circle::Circle(double radius)
{
    this->radius = radius;
}

double Circle::area()
{
    return PI * radius * radius;
}
```

Class declarations give a scope for labels.

- variable names
- function names

Otherwise you need to specify that you want to use it

Namespaces

- Can create an arbitrary scope with a namespace block

```
namespace myNamespace{
    int someVar;
    int anotherVar;
    class someClass{ ... };
    class anotherClass{ ... };
}

myNamespace::someVar = 1;
```

Namespaces

- Mechanism created to reduce naming conflicts
- The standard libraries are in the "std" namespace
- The global namespace is simply "" nothing

Ex.

```
std::cout
::someGlobalVariable
```

Namespaces

```
#include  <iostream>

int main()
{
    std::cout<< "I know how to add!" << std::endl;
    int a, b, sum;
    std::cin >> a;
    std::cin >> b;
    sum = a + b;
    std::cout<< a <<"+"<< b <<"="<< sum << std::endl;
}
```

Namespaces

```
#include  <iostream>

using namespace std;

int main()
{
    cout<< "I know how to add!" << endl;
    int a, b, sum;
    cin >> a;
    cin >> b;
    sum = a + b;
    cout<< a <<"+"<< b <<"="<< sum << endl;
}
```

Namespaces

Can also define each specifically:

```
using std::cout;
```

Watch out for "using namespace ---" lines in .h files

For this course

- do not define namespace explicitly

Today

Introduction to C++

C++ Class structure

Pointers, Briefly

Separate Compilation

Namespaces