C and C++ IV

Spring 2014
Carola Wenk

C and C++

- The C language was originally developed in the 1970s to assist in the implementation of the UNIX operating system. It was designed to be one step above machine language.

- C++ is a superset of C introduced in the early 1980s to add objected-oriented features to C.

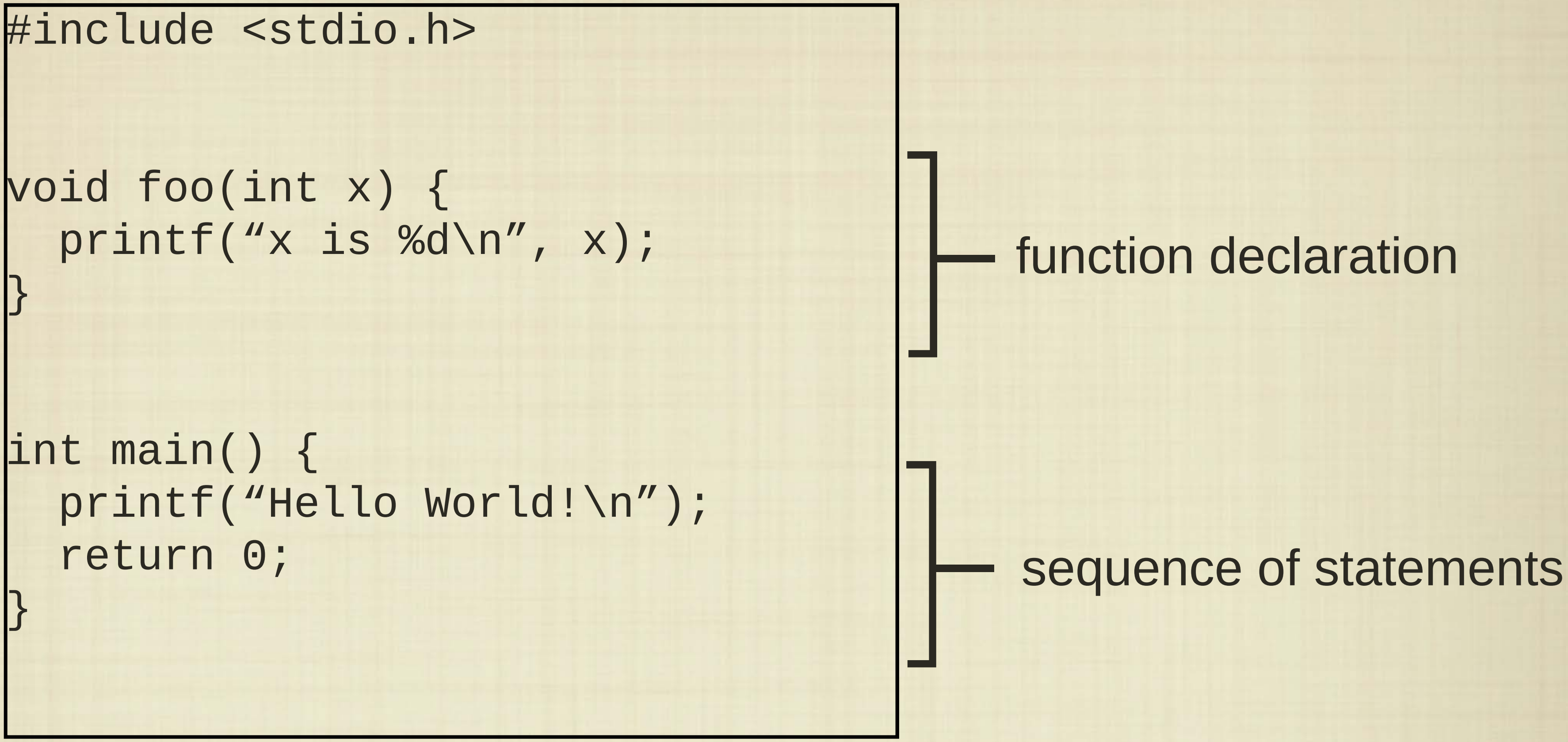
Hello World in C:

```c
#include <stdio.h>

int main() {
     printf("Hello World!!");
     return 0;
}
```

Hello World in C++:

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

int main() {
     cout << "!!!Hello World!!!" << endl;
     return 0;
}
```

C Program Structure



```
#include <stdio.h>


void foo(int x) {
  printf(“x is %d\n”, x);
}


int main() {
  printf(“Hello World!\n”);
  return 0;
}
```

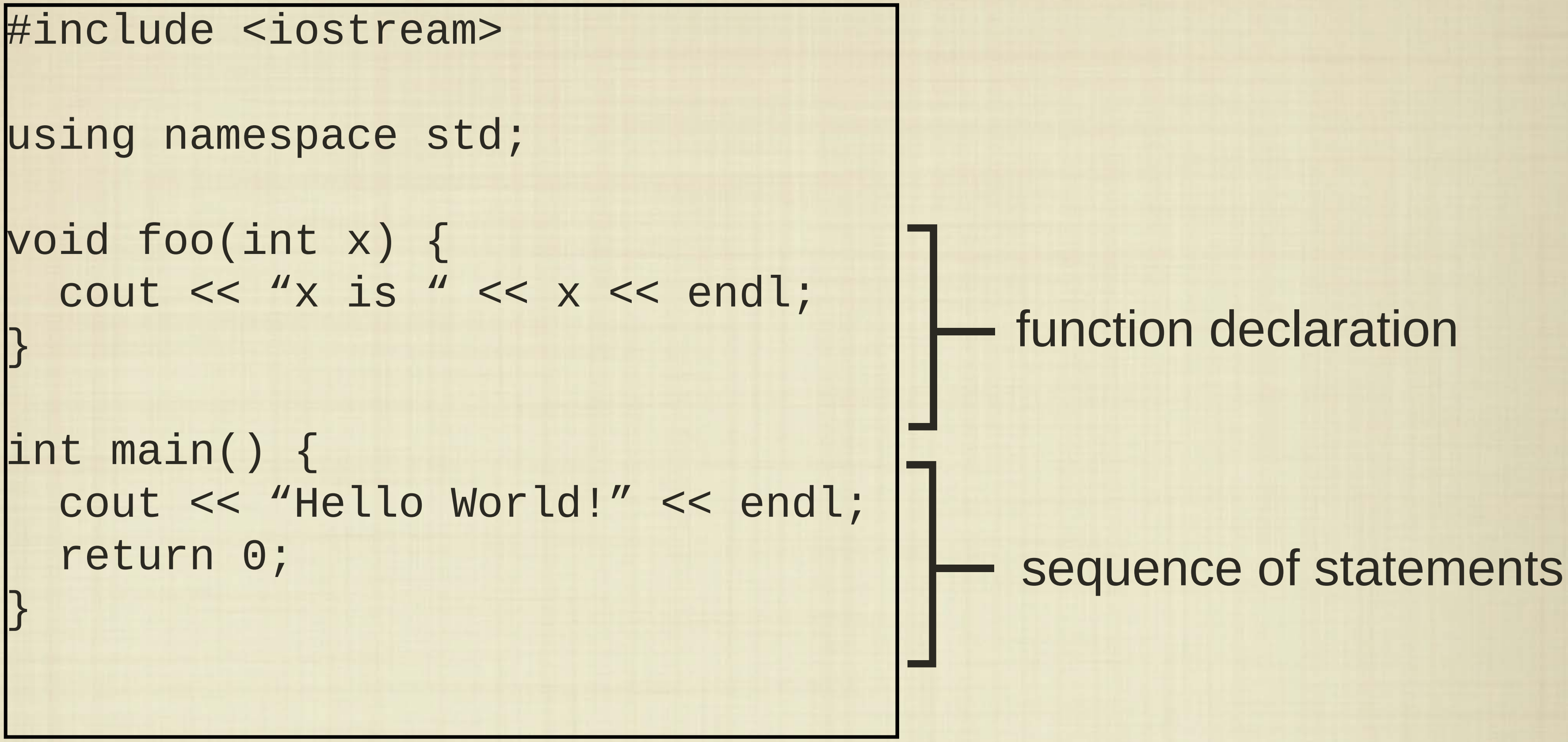
function declaration

sequence of statements

Syntax in C/C++ is very similar to Java, for historical reasons.

However, **not** everything is an object, and programs are initiated from a `main` function.

C++ Program Structure



```
#include <iostream>

using namespace std;

void foo(int x) {
  cout << “x is “ << x << endl;
}

int main() {
  cout << “Hello World!” << endl;
  return 0;
}
```

function declaration

sequence of statements

C is older than C++, and is somewhat more low-level, with different input/output syntax, and no facility to define classes.

C++ is a superset of C, with the ability to define classes.

Namespaces

```
#include <iostream>

using namespace std;

void foo(int x) {
  cout << “x is “ << x << endl;
}

int main() {
  cout << “Hello World!” << endl;
  return 0;
}
```

Namespaces are the analog of packages, and provide scope for library methods. The namespace `std` is where `cout` and `cin` “live”.

Namespaces

```
#include <iostream>



void foo(int x) {
  std::cout << “x is “ << x << std::endl;
}

int main() {
  std::cout << “Hello World!” << std::endl;
  return 0;
}
```

Namespaces are the analog of packages, and provide scope for library methods. The namespace `std` is where `cout` and `cin` “live”.

Namespaces

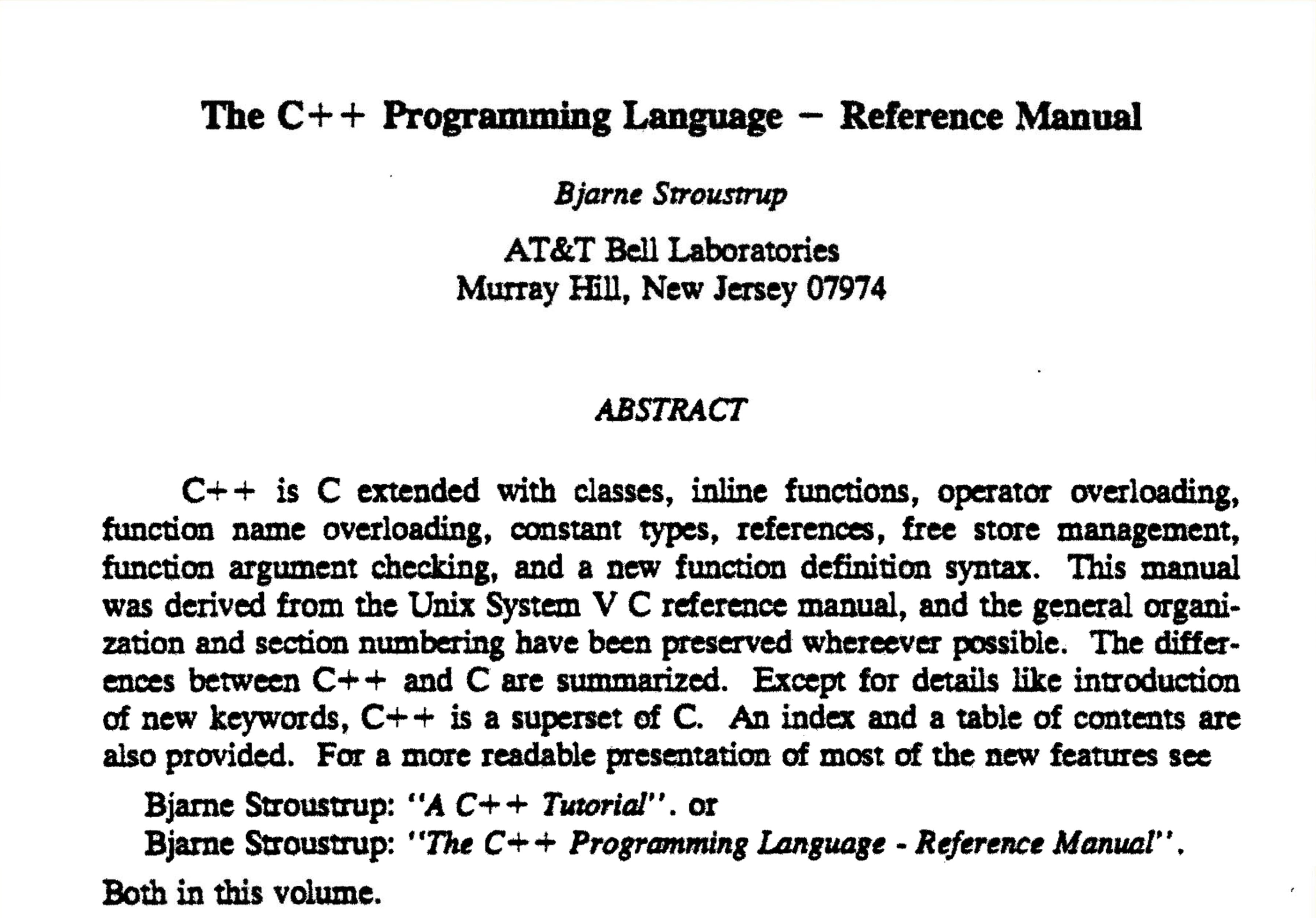
```
#include <iostream>
using namespace std;

  namespace cmps1600 {
    int sleep(){
      return 8;
    }
  }

  namespace summer {
    int sleep(){
      return 12;
    }
  }

int main() {
  cout << “I sleep “ << cmps1600::sleep() << “ hrs. during the semester.”
       << endl;

  cout << “I sleep “ << summer::sleep() << “ during summer.” << endl;
}
```

Namespaces

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

  namespace cmps1600 {
    int sleep(){
      return 8;
    }
  }

  namespace summer {
    int sleep(){
      return 12;
    }
  }

int main() {
  using namespace cmps1600;
  cout << “I sleep “ << sleep() << “ hrs. during the semester.”
       << endl;

  cout << “I sleep “ << summer:: sleep() << “ during summer.” << endl;
}
```

Namespaces

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

  namespace cmps1600 {
    int sleep(){
      return 8;
    }
  }

  namespace summer {
    int sleep(){
      return 12;
    }
  }

int main() {
  { using namespace cmps1600;
    cout << “I sleep “ << sleep() << “ hrs. during the semester.”
         << endl;
  }

  { using namespace summer;
    cout << “I sleep “ << sleep() << “ during summer.” << endl;
  }
}
```



The C++ Programming Language — Reference Manual

Bjarne Stroustrup

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

C++ is C extended with classes, inline functions, operator overloading, function name overloading, constant types, references, free store management, function argument checking, and a new function definition syntax. This manual was derived from the Unix System V C reference manual, and the general organization and section numbering have been preserved whereever possible. The differences between C++ and C are summarized. Except for details like introduction of new keywords, C++ is a superset of C. An index and a table of contents are also provided. For a more readable presentation of most of the new features see

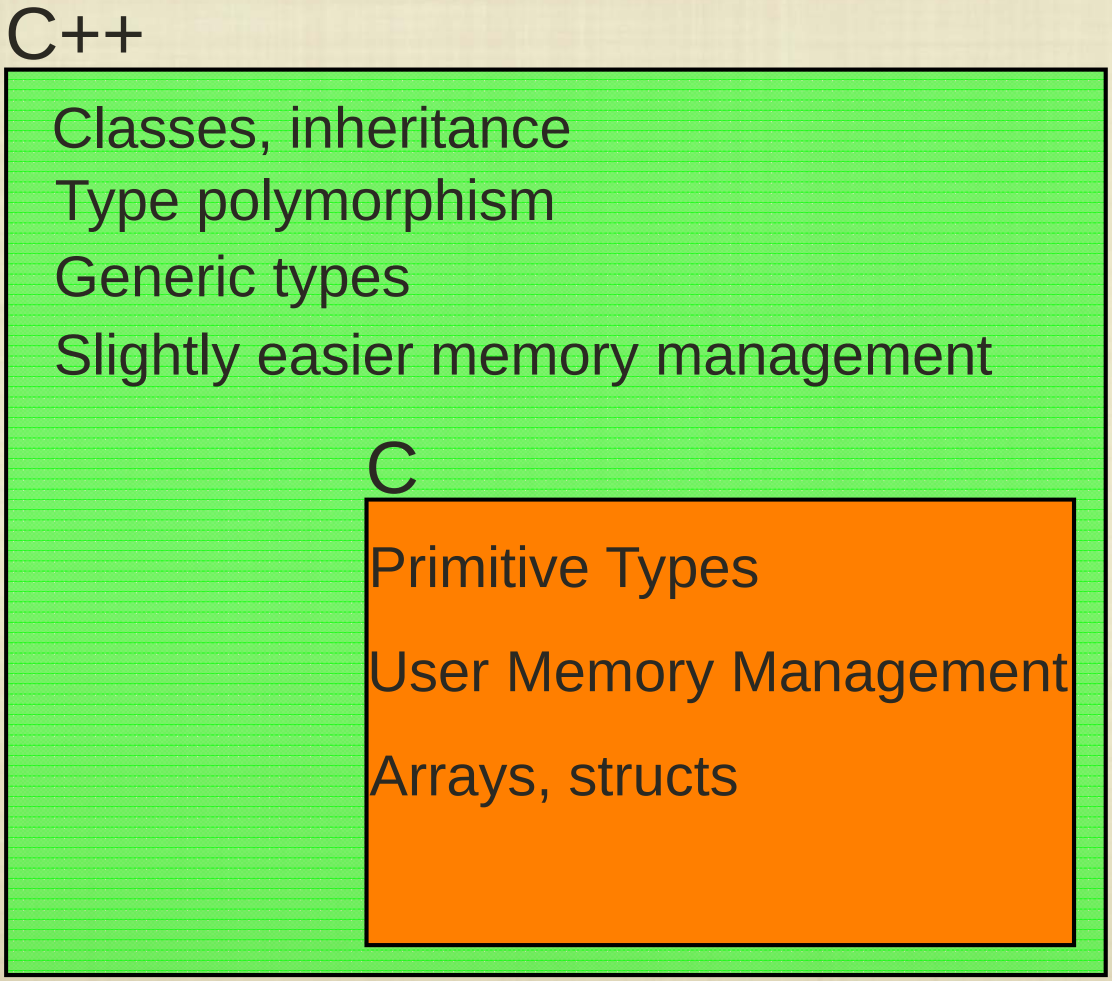
Bjarne Stroustrup: "*A C++ Tutorial*". or
Bjarne Stroustrup: "*The C++ Programming Language - Reference Manual*".

Both in this volume.

[October 1984]

C++

- Motivation (1980’s): C is great, and everyone uses it, so let’s add a bunch of features to it.



C++

Classes, inheritance

Type polymorphism

Generic types

Slightly easier memory management

C

Primitive Types

User Memory Management

Arrays, structs

Object-Oriented Design

- The first object-oriented language was Simula 67; it introduced objects, classes, virtual functions, and garbage collection.

- One of the main goals of Simula was to enable complex discrete event simulations - an event could be defined as a class.

- There are a numerous “pure” and non-pure object-oriented languages: Smalltalk, Eiffel, Scala, Oberon, Java, C#, Objective C, etc.

- Generally speaking, object-oriented design enforces “good habits” of programming large-scale software systems, by localizing functionality.

C++ Class Definitions

private by default

“sections” for access modifiers

instantiation is different

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

class Square{
	double side;
public:
	Square(double s){
	   side = s;
	}
	double area(){
	   return side*side;
	}
	double perimeter(){
	  return 4*side;
	}
};

int main(){
  Square* A = new Square(2.0);
  Square B(3.0);
  cout << “A->area()==“ << A->area() << endl;
  cout << “B.area()==“ << B.area() << endl;
}
```

With the exception of defining a wrapper class, there are only minor differences between Java and C++ class definitions.

C++ Class Definitions

private by default

“sections” for access modifiers (`public`, `private`, `protected`)

instantiation is different

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

class Square{
    double side;
public:
    Square(double s){
        side = s;
    }
    double area(){
        return side*side;
    }
    double perimeter(){
      return 4*side;
    }
};

int main(){
  Square* A = new Square(2.0);
  Square B(3.0);
  cout << “A->area()==“ << A->area() << endl;
  cout << “B.area()==“ << B.area() << endl;
}
```

Notice that references (to objects) are really just pointers as in C. Java hides this distinction for ease of use.

C++ Class Definitions

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

class Square{
     double side;
public:
     Square(double s);
     double area();
     double perimeter(){
       return 4*side;
     }
};

Square::Square(double s){
  this.side = s;
}
double Square::area(){
  return side*side;
}

int main(){
  Square* A = new Square(2.0);
  Square B(3.0);
  cout << “A->area()==“ << A->area() << endl;
  cout << “B.area()==“ << B.area() << endl;
}
```

Can declare members outside of class declaration

Using appropriate “scope” specifiers, we can declare class methods anywhere.

C++ Class Definitions

Square.h

```
#ifndef SQUARE_H
#define SQUARE_H
class Square{
    double side;
public:
    Square(double s);
    double area();
    double perimeter();
};
#endif
```

Square.cpp

```
#include “Square.h”
Square::Square(double s){
   side = s;
}

double Square::area(){
   return side*side;
}

double Square::perimeter(){
   return 4*side;
}
```

main.cpp

```
#include <iostream>
#include “Square.h”
using namespace std;

int main() {
  Square* A = new Square(2.0);
  cout << “A->area()==“ << A->area() << endl;
}
```

Searches in some implementation-dependent path.

Searches in same directory as .cpp file.

Split the class definition into header file and source file, and use the class in another source file.

C++ Class Definitions

Square.h

```
#ifndef SQUARE_H
#define SQUARE_H
class Square{
     double side;
     static int c;
public:
     Square(double s);
     double area();
     double perimeter();
     int count();
};
#endif
```

Square.cpp

```
#include “Square.h”
int Square::c=0;
Square::Square(double s){
   side = s;
   c++;
}
double Square::area(){
   return side*side;
}
double Square::perimeter(){
   return 4*side;
}
int Square::count(){
   return c;
}
```

main.cpp

```
#include <iostream>
#include “Square.h”
using namespace std;

int main() {
  Square* A = new Square(2.0);
  cout << “A->area()=“ << A->area() << endl;
  cout << “A->count()=“ << A->count() << endl;
}
```

Static members behave the same as in Java.

Constructors and Destructors

```
class Buffer {
  int* storage;

public:
  Buffer(int capacity) {
    storage = new int[capacity];
  }

  ...

  ~Buffer(){
    delete []storage;
  }
};
```

Recall that we have to manage allocation and deallocation of data structures: Every class can declare a “destructor” to specify how each instance can free the memory that it has allocated.