SWINBURNE UNIVERSITY OF TECHNOLOGY

COS30008 Data Structures and Patterns

Lecture 1 Introduction to C++

Unit Admin

- **Academic Staff:** Carmen Chai
- **Class Hours:**
  - **Lecture:** Tuesday 10:30am – 12:30am at B511-B512
  - **Lab:** Wednesday 1:30pm – 3:30pm at G518
- **Assessments:**
  - 5 Problem Sets (5% each, 25% total. One will be released every 2 weeks starting from Week 2.)
  - Mid Term Exam (25%)
  - Final Exam (50%)

Unit Admin

- Plagiarism and misconduct
  - Academic misconduct is a form of fraud and intellectual theft and is a serious breach of academic integrity. It covers a range of activities including:
    plagiarism, cheating, failing to comply with examination or assessment rules or directions, engaging in other conduct with a view to gaining unfair or unjustified advantage, committing research misconduct.
  - http://www.swinburne.edu.au/about/leadership-governance/policies-regulations/statutes-regulations/student-academic-misconduct/
- Attendance
  - Attendance is important, it has an indirect impact on your result
- Communication
  - Via email, or email to make an appointment with 24 hours notice

Unit Aims

- How can a given problem be effectively expressed?
- What are suitable data representations for specifying computational processes?
- What is the impact of data and its representation with respect to time and space consumption
- What are the reoccurring structural artefacts in software and how can we identify them in order to facilitate problem solving?

Learning Objectives

- 1.Apply object oriented design and implementation techniques.
- 2.Interpret the tradeoffs and issues involved in the design, implementation, and application of various data structures with respect to a given problem.
- 3. Design, implement, and evaluate software solutions using behavioural, creational, and structural software design patterns.
- 4. Explain the purpose and answer questions about data structures and design patterns that illustrate strengths and weaknesses with respect to resource consumption.
- 5. Assess the impact of data structures on algorithms.
- 6. Analyse algorithm designs and perform best-, average-, and worst- case analysis.

Overview

- The following gives a tentative list of topics not necessarily in the order in which they will be covered in the subject
  - Introduction to C++
  - Sets, Arrays, Indexers, and Iterators
  - Basic Data Structures and Pattern
  - Abstract Data Types and Data Representation
  - One-Dimensional Data Structure
  - Hierarchical Data Structures
  - Algorithmic Patterns and Problem Solvers

Why do we need data structures?

- Programs are comprised of two things: **data** and algorithms.
  - The algorithms describe the way the **data** is to be transformed.
  - The reason for learning about **data structures** is because adding **structure** to our **data** can make the algorithms much simpler, easier to maintain, and often faster
- **Data structure** is a particular way of storing and organizing information in a computer so that it can be retrieved and used most productively.
- A data structure is a **specialized** format for organizing and storing data.
- General data structure types include the array, the file, the record, the table, the tree, and so on. Any data structure is designed to organize data to suit a specific purpose so that it can be accessed and worked with in appropriate ways.

A brief introduction to C++

- We need to know more than just Java.
- C++ is highly efficient and provides a better match to implement low-level software layers like device controllers or networking protocols.
- C++ is being widely used to develop commercial applications and is a the centre of operating system and modern game development
- Memory is tangible in C++ and we can, therefore, study the effects of design decisions on memory management more directly.

Core Properties of Programming Languages

- Programming languages provide us with a framework to organize computational complexity in our own minds.
- Programming languages offer us the means by which we communicate our understanding about a computerized problem solution.

What is C++

- C++ is a general-purpose, high-level programming language with low-level features
- Bjarne Stroustrup developed C++ (C with Classes) in 1983 at Bell Labs as an enhancement to the C programming language.
- A first C++ programming language standard was ratified in 1998 as ISO/IEC 14882:1998. The current extending version is ISO/IEC 14882:2011 (informally known as C++11).

Design Philosophy of C++

- C++ is a hybrid, statically-typed, general-purpose language that is as efficient and portable as C
- C++ directly supports multiple programming styles like procedural programming, object-oriented programming, or generic programming.
- C++ gives the programmer choice, even if this makes it possible for the programmer to choose incorrectly!
- C++ avoids features that are platform specific or not general purpose, but is itself platform-dependent
- C++ does not incur overhead for features that are not used
    - C++ functions without an integrated and sophisticated programming environment.

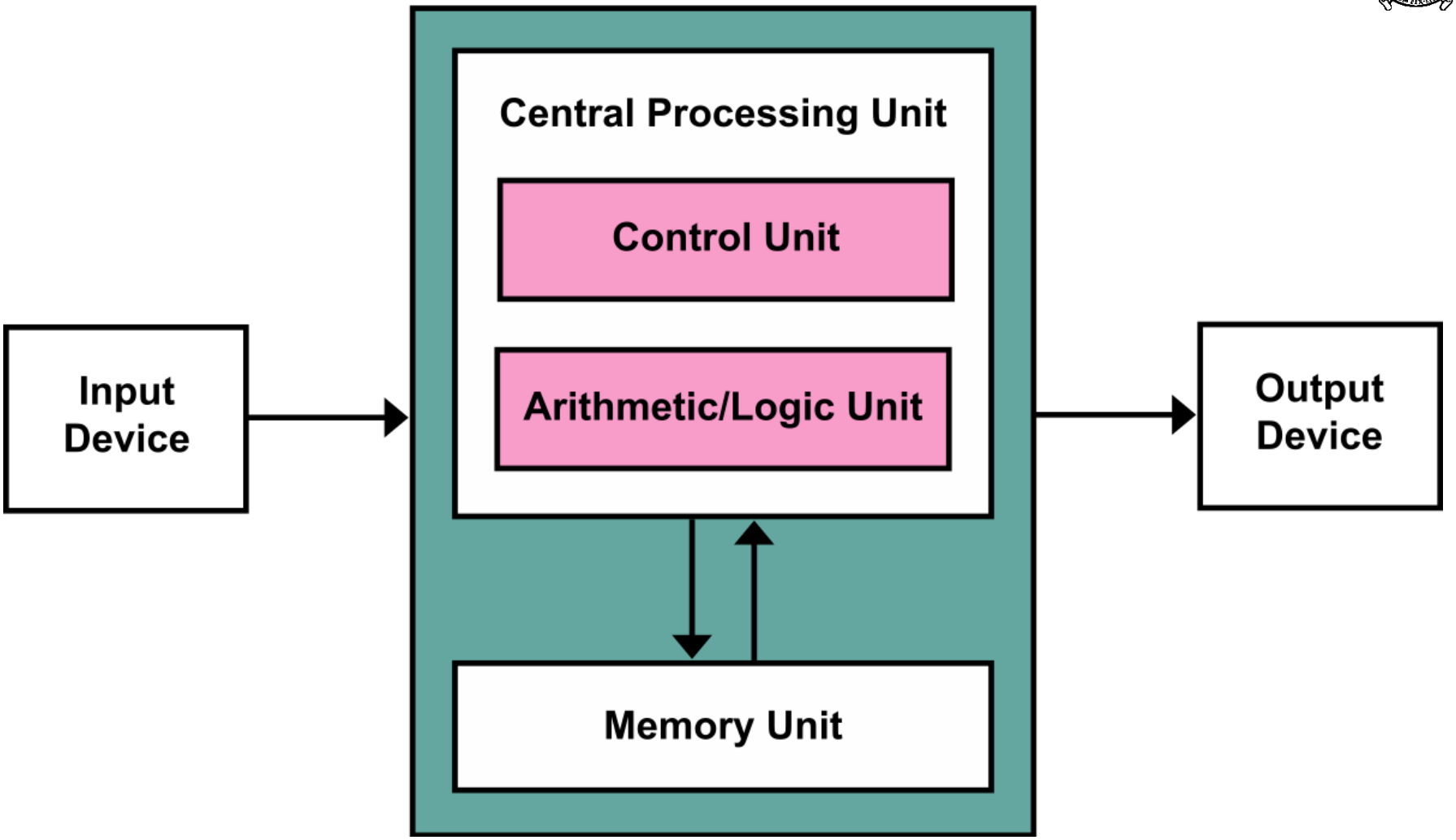
C++ Paradigms

- C++ is a multi-paradigm language
- C++ provides natural support for
  - the imperative paradigm and
  - the object-oriented paradigm.

Imperative Programming

- This is the oldest style of programming, in which the algorithm for the computation is expressed explicitly in terms of instructions such as assignments, tests, branching and so on.
- Execution of the algorithm requires data values to be held in variables which the program can access and modify.
- Imperative programming corresponds naturally to the earliest, basic and still used model for the architecture of the computer, the von Neumann model.

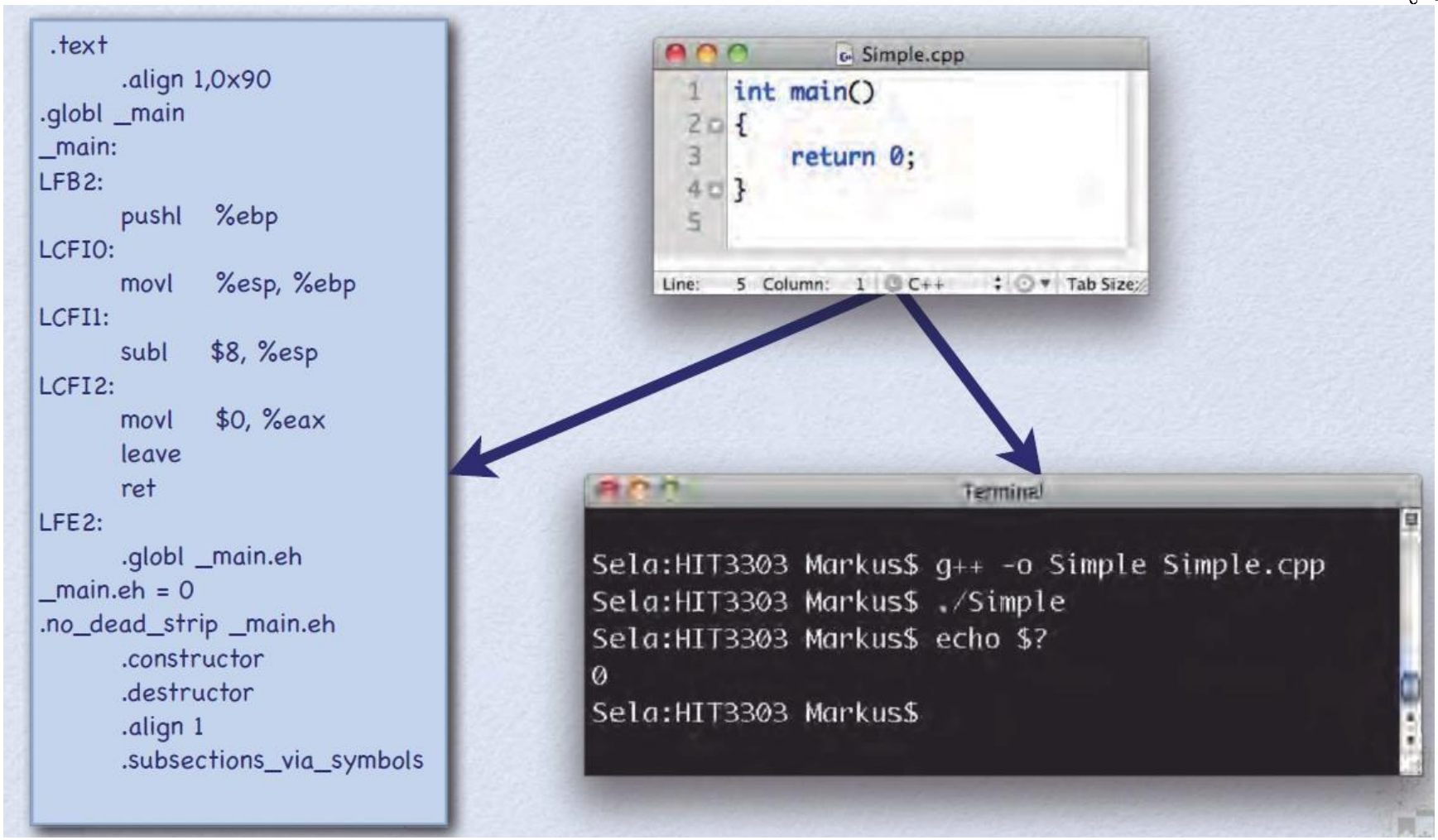
The von Neumann Architecture



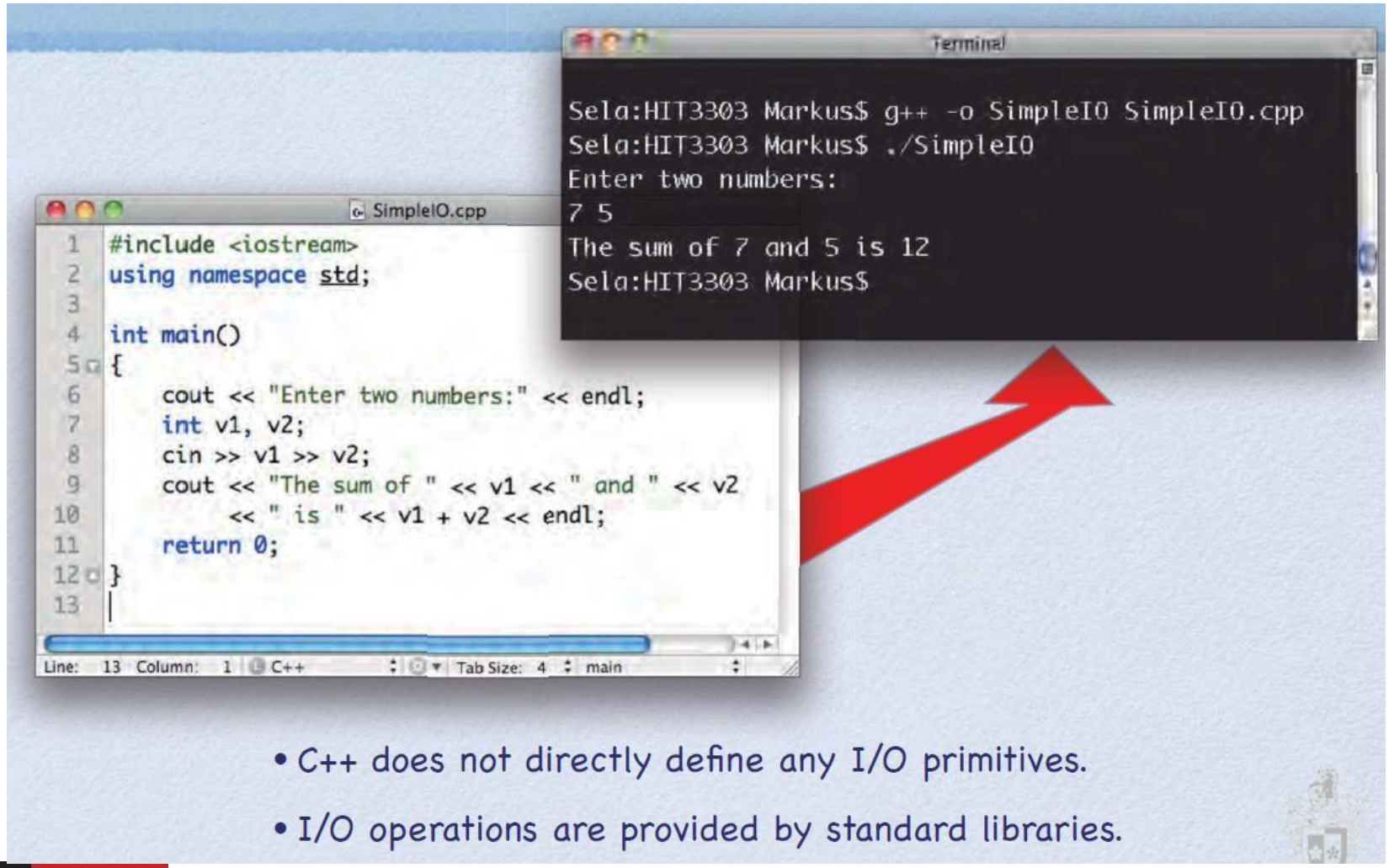
Object-Oriented Programming

- In general, object-oriented languages are based on the concepts of class and inheritance, which may be compared to those of type and variable respectively in a language like Pascal and C.
- A class describes the characteristics common to all its instances, in a form similar to the record of Pascal (structures in C), and thus defines a set of fields.
- In object-oriented programming, instead of applying global procedures or functions to variables, we invoke the methods associated with the instances (i.e., objects), an action called “message passing.”
    - The basic concept of inheritance is used to derive new classes from existing ones by modifying or extending the inherited class(es).

The Simplest Possible C++ Program

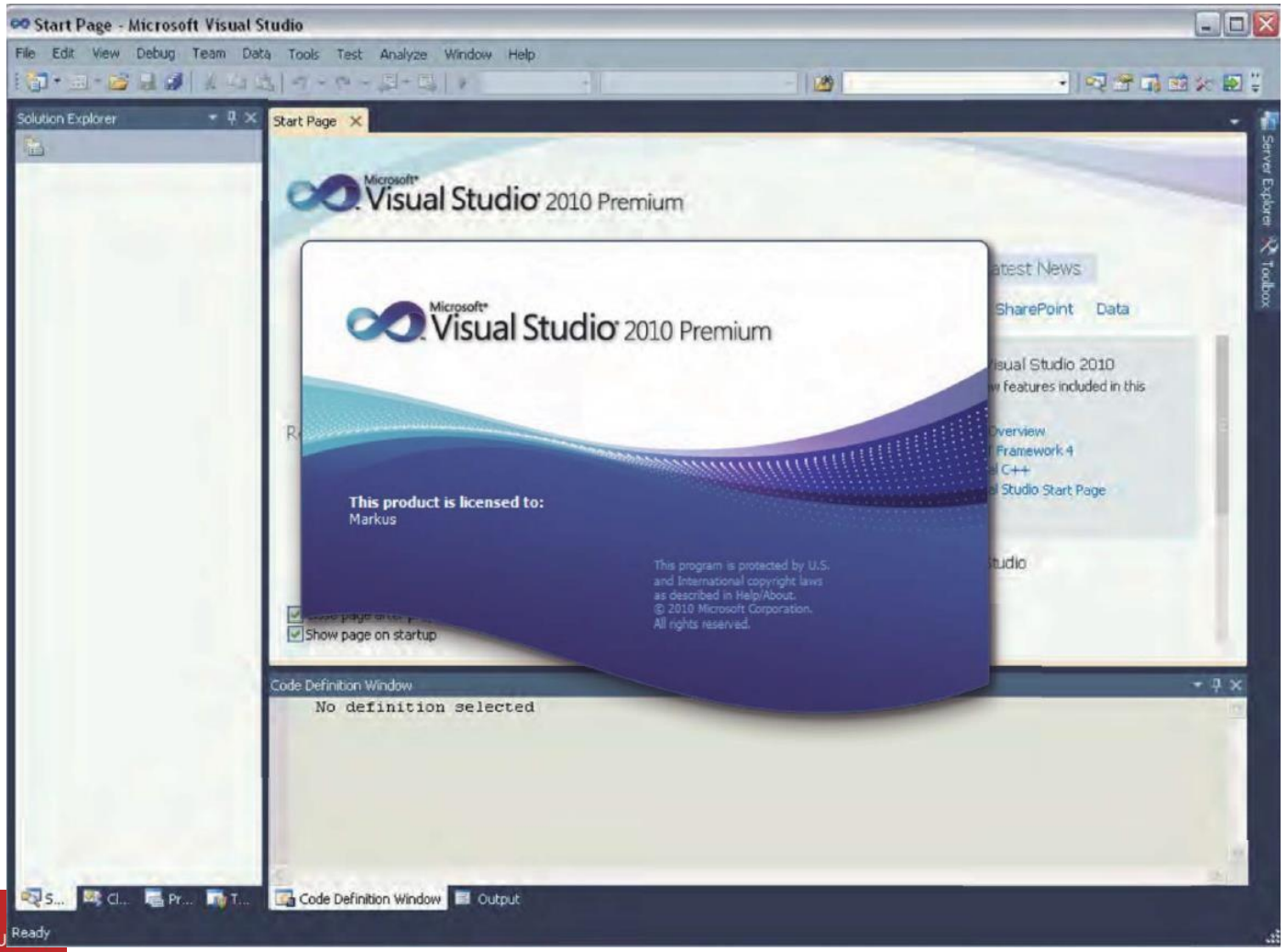


Let's make the program more responsive!



IDE to use for this unit

- Microsoft Visual Studio 2010/2012



Simple Class Example in C++

URL: https://www.youtube.com/watch?v=PimQZQRf-88

BookStore Example

- Before we can write a program in a new language, we need to know some of its basic features. C++ is no exception.
- The example class in this lecture is the book store program requires us to
  - define variables
  - perform input and output
  - define a data structure (aka a class) to hold the data we are managing
  - test whether two items (aka objects) denote the same value
  - write control code to process data

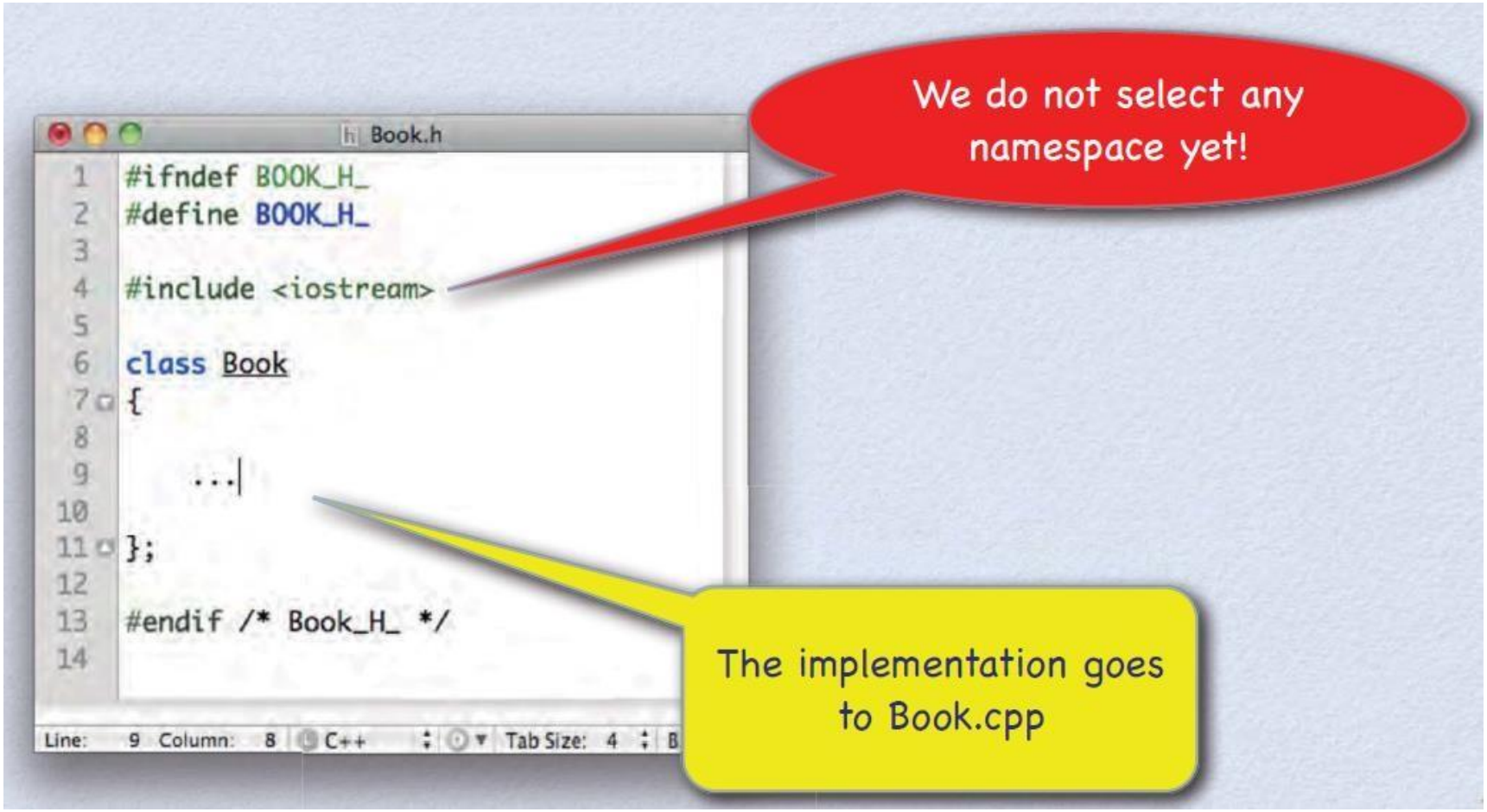
C++ Code Organization

- Classes are defined in include files (i.e., .h).
- Class members are implemented in source files (i.e., .cpp).
- There are exceptions (as usual), when working with templates.

The Structure of a Class

```
class X
{
private:
// private members

protected:
// protected members

public:
// public members
};
```

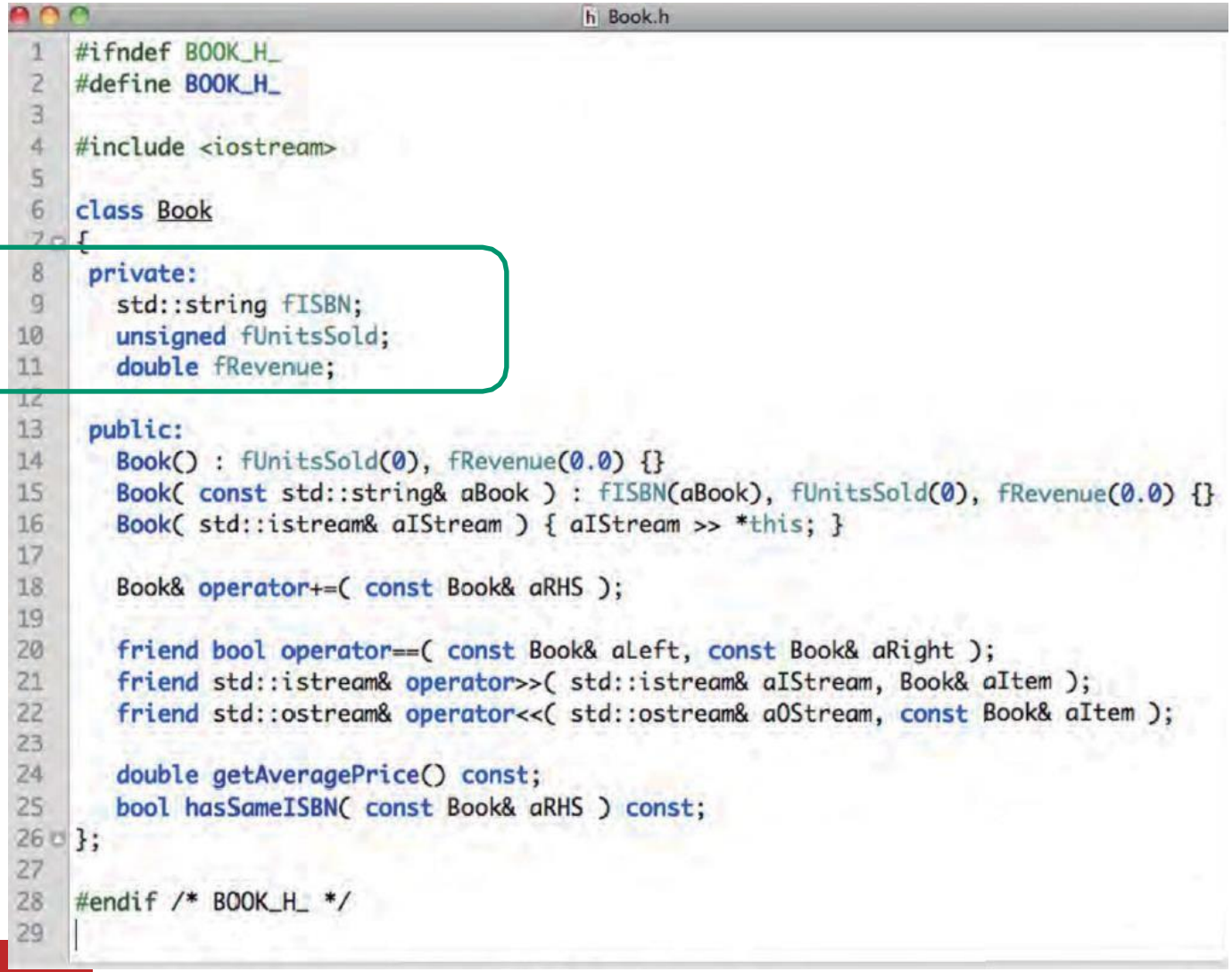
Book.h


Book.h
1 #ifndef BOOK_H_
2 #define BOOK_H_
3
4 #include <iostream>
5
6 class Book
7 {
8
9 ...
10
11 };
12
13 #endif /* Book_H_ */
14
Line: 9 Column: 8 C++ Tab Size: 4
We do not select any namespace yet!
The implementation goes to Book.cpp

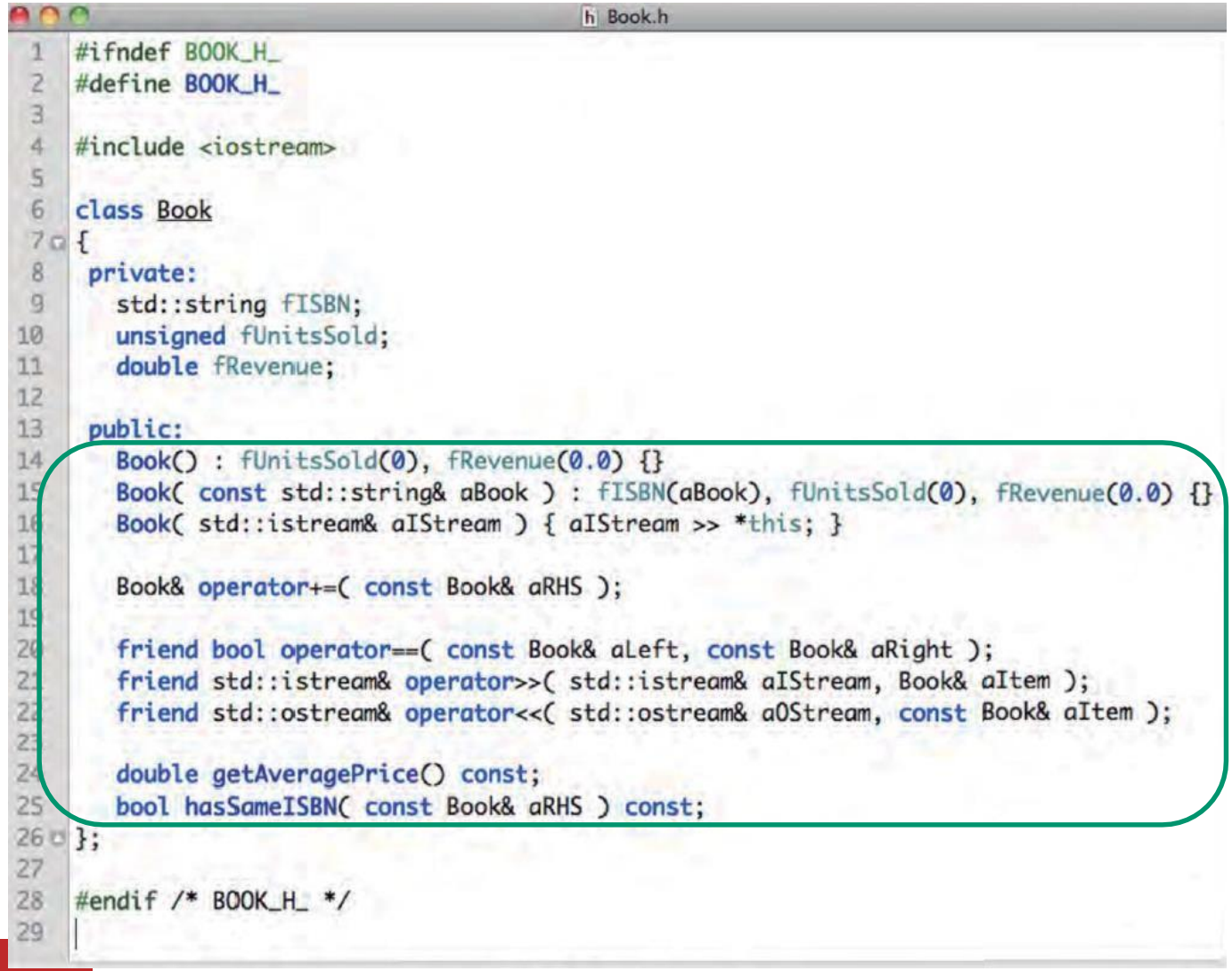
Access Modifiers

- **private:**
  Private members can be accessed only within the class in which they are declared. Usually class variables are declared here.
- **protected:**
  Protected members can be accessed within the class in which they are declared and within derived classes.
- **public:**
  Public members can be accessed anywhere, including outside of the class itself. Usually methods constructors, accessors, operator overloads and other functions used with the class are declared here.

Book Class - Private Members



Book Class - Public Members



Standard Boilerplate Code

- In computer programming, **boilerplate code** or **boilerplate** refers to sections of code that have to be included in many places with little or no alteration. It is often used when referring to languages that are considered *verbose*, i.e. the programmer must write a lot of code to do minimal jobs.

The #include guard

```
#ifndef HEADER_H_
#define HEADER_H_

  /* Body of Header */

#endif /* HEADER_H_ */
```

- The #include guard sets up, and checks a global flag to tell the compiler whether the file header.h has already been included. As many interdepending files may be involved in the compilation of a module, this avoids processing the same header multiple times (which would lead to errors due to multiple definitions with the same name).

C++ Class Specification

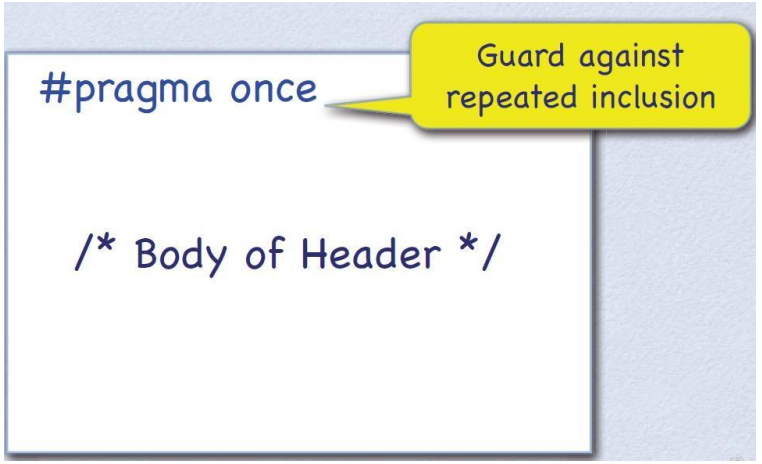
Use of #include guard in the Book Class

Book.h

```
#ifndef BOOK_H_
#define BOOK_H_

#include <iostream>

class Book
{
 private:
   std::string fISBN;
   unsigned fUnitsSold;
   double fRevenue;

 public:
   Book() : fUnitsSold(0), fRevenue(0.0) {}
   Book( const std::string& aBook ) : fISBN(aBook), fUnitsSold(0), fRevenue(0.0) {}
   Book( std::istream& aIStream ) { aIStream >> *this; }

   Book& operator+=( const Book& aRHS );

   friend bool operator==( const Book& aLeft, const Book& aRight );
   friend std::istream& operator>>( std::istream& aIStream, Book& aItem );
   friend std::ostream& operator<<( std::ostream& aOStream, const Book& aItem );

   double getAveragePrice() const;
   bool hasSameISBN( const Book& aRHS ) const;
};

#endif /* BOOK_H_ */
```

#pragma once (Visual Studio)

- "#pragma once" does the same thing as the #include guard **but is not standard**. It is a widespread (but not universal) extension, which can be used (1)If your portability concerns are limited, and (2) If you can be sure that all of your include files are always on a local disk.

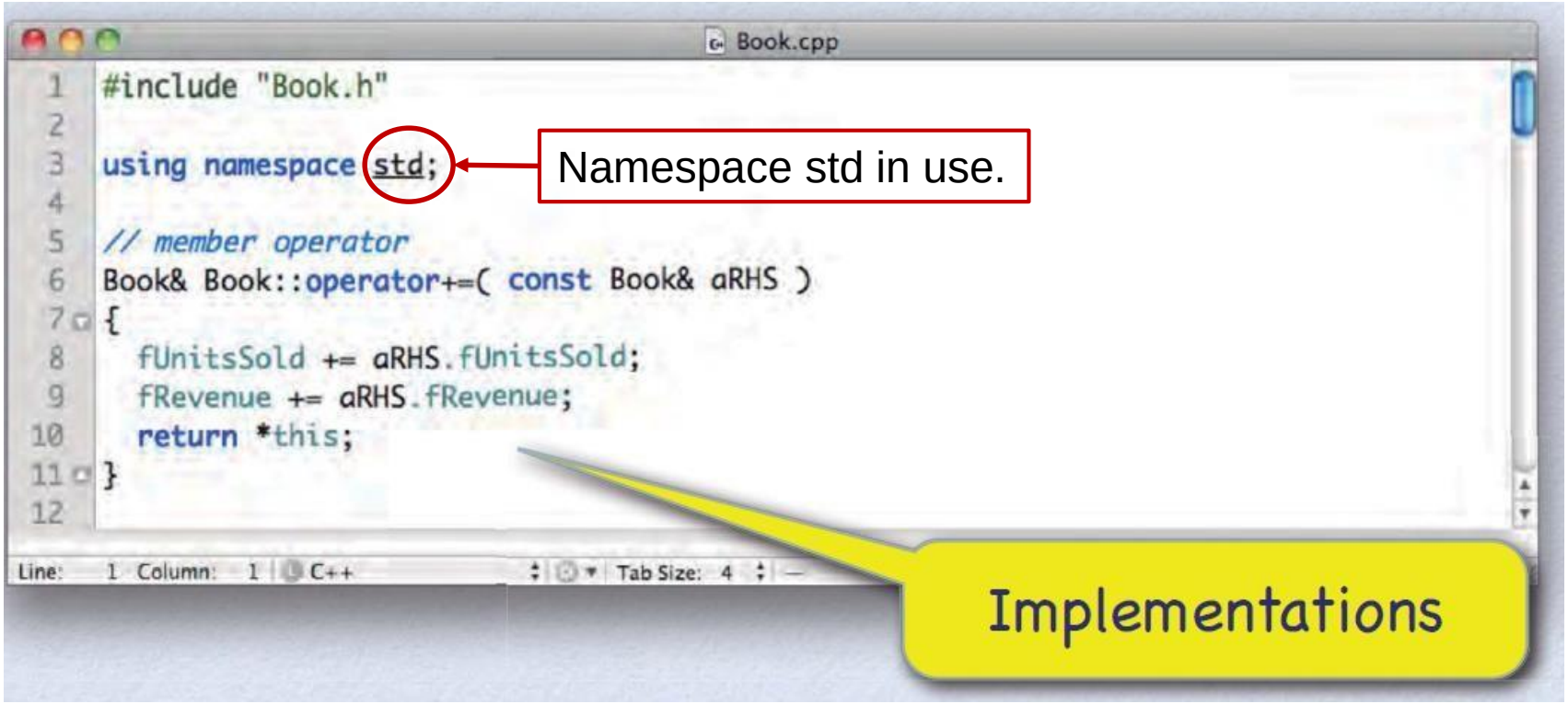


  - It was considered for standardization, but rejected because it cannot be implemented reliably.
    - The problems occur when you have files accessible through several different remote mounts.

namespace

- A namespace is a declarative region that provides a scope to the identifiers (the names of types, functions, variables, etc) inside it.
- Namespaces are used to organize code into logical groups and to prevent name collisions that can occur especially when your code base includes multiple libraries.
- **std** is an abbreviation of standard. **std** is the standard **namespace**. cout, cin and a lot of other things are defined in it. (This means that without the use of the std namespace, you need to call use **`std::`**`cout` and **`std::`**`cin to call them`.)

Book.cpp



This shows the implementation of the operator overload of += for the Book class.

Constructors

- **Constructors** are executed automatically whenever a new object is created.
- A **constructor** is a kind of member function that initializes an instance of its class.
- A **constructor** has the same name as the class and no return value.
- Constructors may be overloaded.
- The concrete constructor arguments determine which constructor to use.
  - A **constructor** can have any number of parameters and a class may have any number of overloaded **constructors**

Constructor Initializer

- A constructor initializer is a comma separated list of member initializers, which is declared between the signature of the constructor and its body.

```
c:\Users\pzheng\Desktop\DSP.PS3\Mark\initi\Debug\initi.exe
The volume is 10584
The volume is 125
Press any key to continue . . .
```

```
1  #include<iostream>
2
3  using namespace std;
4
5  class Box {
6      int m_width;
7      int m_length;
8      int m_height;
9
10 public:
11     Box(int width, int length, int height)
12         : m_width(width), m_length(length), m_height(height){}
13     Box(): m_width(5), m_length(5), m_height(5){}
14
15     int Volume() {return m_width * m_length * m_height; }
16 };
17
18 int main(){
19     Box b(42, 21, 12), a;
20     cout << "The volume is " << b.Volume()<<endl;
21     cout << "The volume is " << a.Volume()<<endl;
22
23     system("pause");
24     return 0;
25 }
```

Class Book - Constructors

Book.h

```
1  #ifndef BOOK_H_
2  #define BOOK_H_
3
4  #include <iostream>
5
6  class Book
7  {
8   private:
9     std::string fISBN;
10    unsigned fUnitsSold;
11    double fRevenue;
12
13  public:
14    Book() : fUnitsSold(0), fRevenue(0.0) {}
15    Book( const std::string& aBook ) : fISBN(aBook), fUnitsSold(0), fRevenue(0.0) {}
16    Book( std::istream& aIStream ) { aIStream >> *this; }
17
18    Book& operator+=( const Book& aRHS );
19
20    friend bool operator==( const Book& aLeft, const Book& aRight );
21    friend std::istream& operator>>( std::istream& aIStream, Book& aItem );
22    friend std::ostream& operator<<( std::ostream& aOStream, const Book& aItem );
23
24    double getAveragePrice() const;
25    bool hasSameISBN( const Book& aRHS ) const;
26 };
27
28 #endif /* BOOK_H_ */
29
```

Default Constructor

- A default constructor is one that does not take any arguments.
- The compiler will synthesize a default constructor, when no other constructors have been specified.
- If some data members have built-in or compound types, then the class should not rely on the synthesized default constructor!

Friends

- Friends are allowed to access private members of classes
- A class declares its friends explicitly
- Friends enable uncontrolled access to members
- The friend mechanism induces a particular programming (C++) style.
- The friend mechanism is not object-oriented.
- I/O depends on the friend mechanism.

The Friend Mechanism

- Friends are self-contained procedures (or functions) that do not belong to a specific class, but have access to the members of a class, when the class declares those procedures as friends.

Class Book - The Friends

```
6  class Book
7  {
8   private:
9     std::string fISBN;
10    unsigned fUnitsSold;
11    double fRevenue;
12
13  public:
14    Book() : fUnitsSold(0), fRevenue(0.0) {}
15    Book( const std::string& aBook ) : fISBN(aBook), fUnitsSold(0), fRevenue(0.0) {}
16    Book( std::istream& aIStream ) { aIStream >> *this; }
17
18    Book& operator+=( const Book& aRHS );
19
20    friend bool operator==( const Book& aLeft, const Book& aRight );
21    friend std::istream& operator>>( std::istream& aIStream, Book& aItem );
22    friend std::ostream& operator<<( std::ostream& aOStream, const Book& aItem );
23
```

- 'friend' is used to make the overload operators accessible globally. The overload operators shown here are (1) == which returns bool, (2) >> which returns istream, and (3) << which returns ostream.
  - These are defined when required because these operators are not made to be used with the Book type.
  - After we define them we are able to for example, use the == operator like Book1 == Book2;

End of Introduction to C++ Part 1

Operator Overloading and more basics of C++ will be covered in the lab class and the next lecture..