CS 2212 Intermediate Programming C++

CHAPTER 13.4 TO END –STRUCTURES AND CLASSES IN C++

Seeding the rand() funtion

#include <time.h> //ctime for C++

…

srand(time(0));

Then rand() will give you unique values.

Objectives

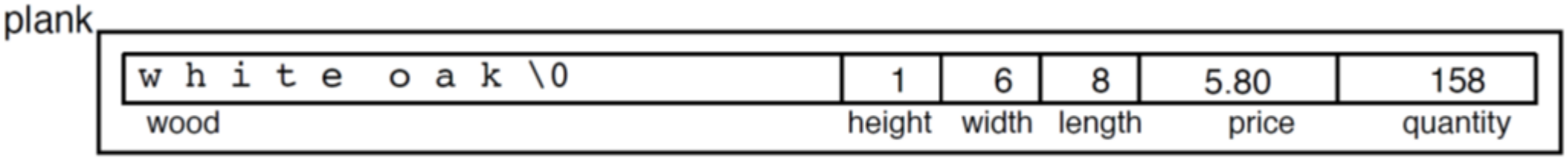
Addressing Course Objective 4: Use pointers, strings, arrays, structures, and classes appropriately in programs.

1. Review Structures in C and compare to Objects of a Class in C++.
2. Be able to define, evaluate appropriateness of use and use an class in C++.
3. Be able to combine class objects with each other and with arrays to represent arbitrarily complex hierarchical objects.
4. Be able to define, evaluate appropriateness of use and use the basic operations on objects .
5. Be able to design classes that meet the design principles of classes.

C Structure is very important

A **C structure** is **much simpler than a class in C++**.

- In C, a structure is a type can contain data of different types.
- You can create variables or arrays of structure or put them in files.
- Structures are a building block for some of the more sophisticated things we have in operating systems and other places.
- You can create **arrays of structures** or put them in files.



C Structure compared to a Class in C++

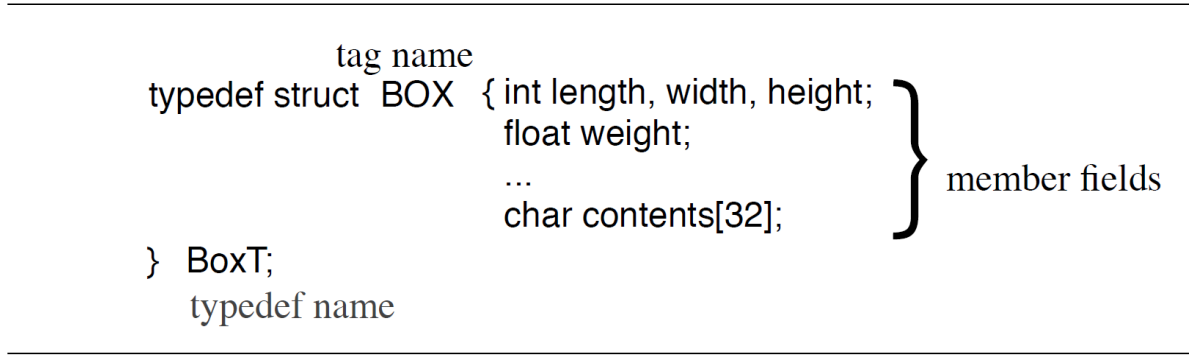
C++ is an Object-Oriented language. **A class is a lot more than a structure.**

- A class allows **data to be encapsulated (hidden)** within the class object by making it **private**. Data Hiding is a Cybersecurity First Principle.
- The class contains the public and private **methods** that act on the object's data.
- **An object takes care if its own business.**
- You can **create arrays of objects** or put them in files.

Structures

MIXED TYPES OF VARIABLES BUNDLED TOGETHER IN A TYPE

C - Structured type – data only



3. Modern syntax for a C `struct` declaration.

New types. A new type name can be defined using typedef and struct, and may be used like a built-in type name anywhere a built-in type name may be used. Important applications are to

1. Declare a structured variable.
2. Create an array of structures.
3. Declare the type of a function parameter.
4. Declare the type of a function return type.

C - Structure Declaration

A `struct` type specification creates a type that can be used to declare variables. The `typedef` declaration names this new type.

```
typedef struct LUMBER {          // Type for a piece of lumber.
    char wood[16];               // Type of wood.
    short int height;            // In inches.
    short int width;             // In inches.
    short int length;            // In feet.
    float price;                 // Per board, in dollars.
    int quantity;                // Number of items in stock.
} LumberT;
```

The members of `LumberT` will be laid out in memory in this order. Depending on your machine architecture, padding bytes might be inserted between member fields.

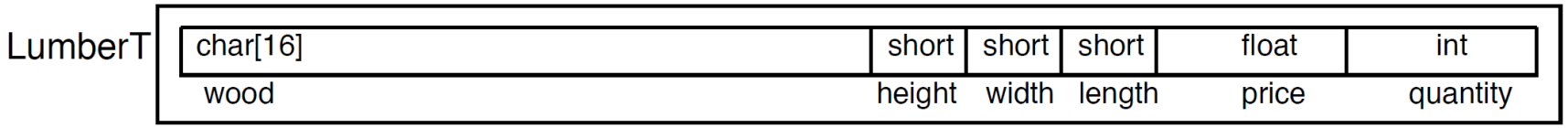
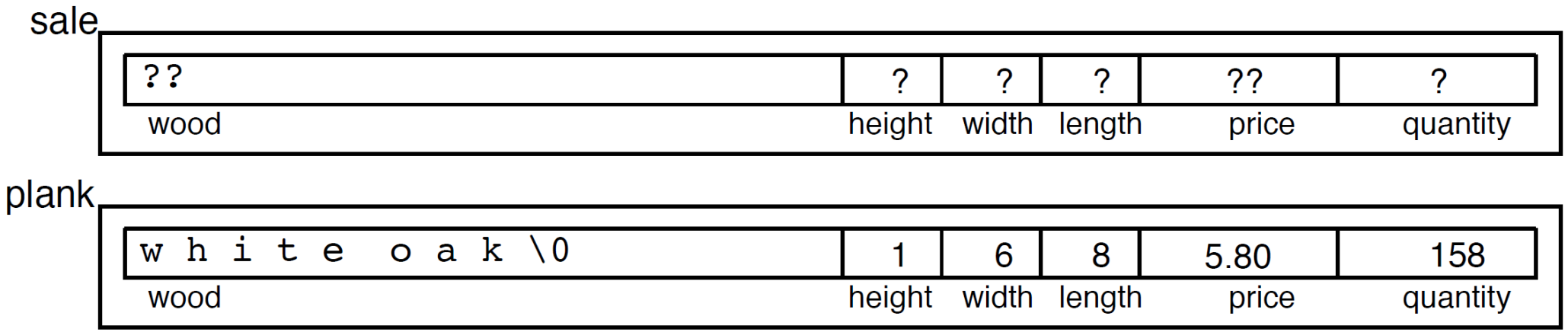


Figure 13.9. A `struct` type declaration in C.

In C - Using a Structured Type

```
LumberT sale;
LumberT plank = { "white oak", 1, 6, 8, 5.80, 158 };
int bytes = sizeof (LumberT);
```



In C – Declaring an Array of Structured Type

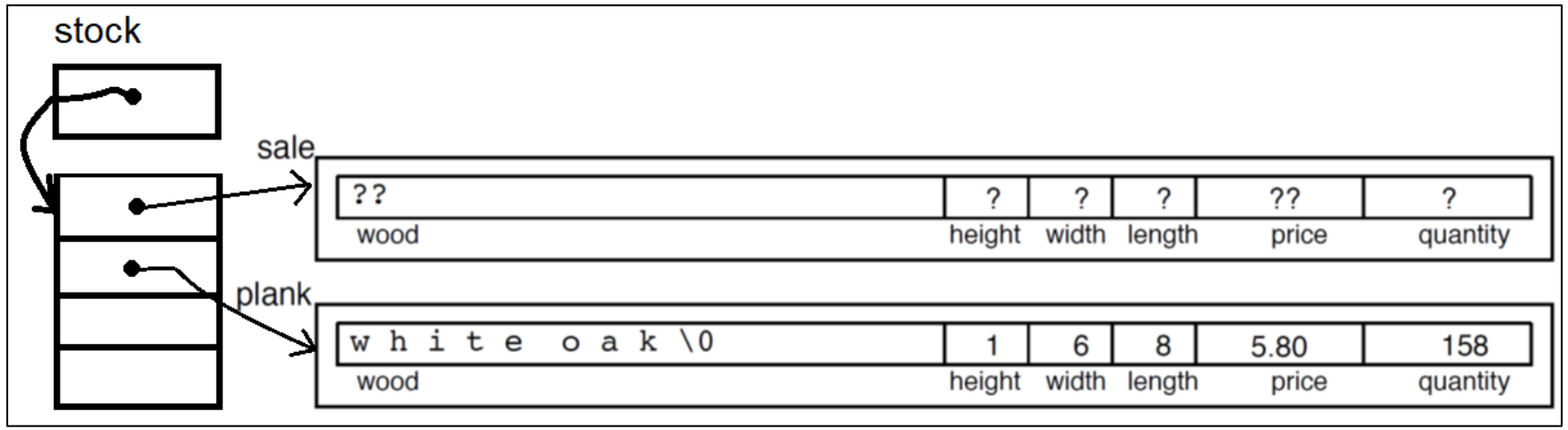
```
LumberT sale;

LumberT plank = {"white oak", 1, 6, 8, 5.80, 158};

LumberT stock[4] ;

stock[0] = sale;

stock[1] = plank;
```



Operations of Structures in C

1. Find the size of the structure.
2. Set a pointer's value to the address of a structure.
3. Access one member of a structure.
4. Use assignment to copy the contents of a structure.
5. Return a structure as the result of a function.
6. Pass a structure as an argument to a function, by value or address.
7. Include structures in larger compound objects, such as an array of structures.
8. Access one element in an array of structures.
9. Access one member of one structure in an array of structures.

Accessing Data members of the Structure in C

Access one member of a structure.

The **members of a structure** may be accessed either directly or through a pointer.

The **dot (period) operator** is used for direct access.

- Write the variable name followed by a dot, followed by the name of the desired member.
- For example, to access the length member of the variable plank, we write

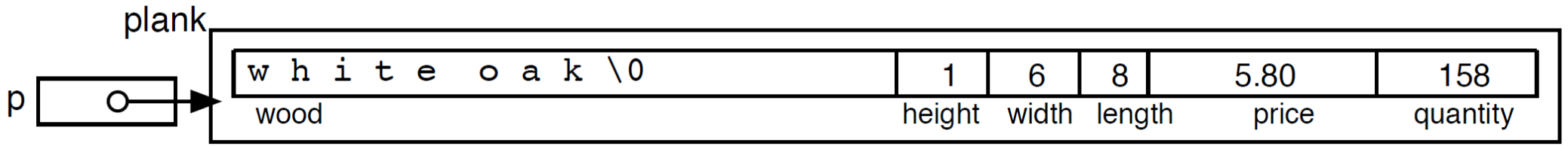
```
plank.length
```

- Once referenced, the member may be used anywhere a simple variable of that type is allowed.

Setting a Pointer to the Structure in C

```
LumberT* p = &plank;
```

```
p = & plank;
```



Accessing a Structure via A Pointer in C

The **arrow operator** is used to **access the members** of a structure **through a pointer**.

We write the pointer name followed by -> (arrow) and the member name.

The **arrow operator** is a **hyphen (aka dash) followed by an angle bracket**.

- For example, since the pointer p is pointing at plank, we can use p to access one member of plank by writing

```
p->price
```

Printing a Data Member in C

```
printf( " %s price is $%.2f\n", plank.wood, plank.price );   // direct access
printf( " %s in stock:  %i\n", p->wood, p->quantity );       // indirect access
```

Figure 13.12. Access one member of a structure.

Classes in C++

THERE ARE NO CLASSES IN C

C++ is an Object Oriented Language and Supports Secure Programming Methods

- **Secure programming** is part of **Computer Science Professionalism.**
- OOP and Secure Programming requires that you follow the Cybersecurity First Principles – **Data Hiding, Simplicity and Modularity.**
  - **Using classes** simplifies coding because **related** data members and functions are **organized into a meaningful module**.
  - Using classes means you are putting source code in **more than one program module, each one with its own responsibilities and groups of data**.
  - The class (Model Class) handles one object of the data at a time. It houses the data and the methods to create and process that data.
- OOP and Secure Programming also require that you follow the Cybersecurity First Principle – **Data Hiding**.
  - Using classes supports **protecting the data (encapsulation)** by making it **private**.
  - Class methods (functions inside the class) can see and mutate the data members, but **outside functions cannot use methods declared as private**.
  - When objects are passed as parameters, the **const** keyword prevents modification by the function that is using the object, **enhancing protection of the data**.
- This requires a lot of thought and planning to design a system that can work in this way.

Defining a Class in C++

To define a class in C++, you must **create two files** (File- New-Class in Eclipse or choose class in Xcode):

- This is your **Model Class**.
- A **header** file (<ClassName>.**hpp**): e.g. `LumberT.hpp`
  - Header files (.hpp) contain only class, enum, and other typedef declarations.
  - The **data members of the class** are declared, but the header file does not contain declarations of variables or arrays for use in a program.
  - The class **method (like functions) prototypes go** here, but they are normally defined in a separate code file (.cpp).
- A **class source code file** (< ClassName >.cpp): e.g. `LumberT.cpp`
  - The class source code file should contain your **Constructor declarations.**
  - The **class's method declarations** also go in its source code file.
  - You must **include the Class header file in your class source code file .** e.g**. #include "LumberT.hpp"**
- The class header and code files **DO NOT contain a main() function**, there can only be ONE main() in each project.

Syntax for Defining a Class – you must follow naming rules

The keyword **`class`** is used to define the class.

Class names START with an **UPPER CASE LETTER**.

- E.g. To name it Lumber use

```
class Lumber { …
```

An **object** is an instance of a class – one element of that kind.

```
Lumber plank; //an object of the Lumber class
```

Object and function names all start with **lower case letters**.

Both use **camelCase** if there is more than one word in the name.

Examples of **Class Declaration** - goes in the Header File

Lumber: Type of wood, dimensions, price per board, quantity in stock.

```
class Lumber { …
```

ArtWork: Artist, Title, Year, Media, Genre, Location, Special Instructions.

```
class ArtWork { …
```

BaseBallTicket: Date of Game, Sections, Row, Seat, availability, price.

```
class BaseBallTicket { …
```

Design Principles for Model Class - **Data**

1. A class **protects its members**. It **keeps them private** and ensures that the data stored in an object is internally consistent and meaningful at all times.

2. A class should be the **expert on its own data members (variables that store the data of the class)**.

3. **Keep data member names short and simple**. If the names of your data members are long, code is more difficult to write, read, and debug.

4. **Good Housekeeping**. With dynamic allocation, new objects should be created by the class that will contain them, and that class is responsible for deleting them when they are no longer needed. Important later when you you write code where you are handling memory allocation directly (if you allocate (malloc), your need to deallocate (delete)).

Design principles of Model Class - **Methods**

1. **Implement all the methods** (functions) needed to act on an object in the model class.
   - Don't try to do things in the wrong class.
   - The methods that belong in the model class are those that **deal with one class instance**.
   - **Delegate** the activity to the **expert – the model class does this because it has the data**.
   - E.g. The right place to process a deposit and **update the balance is where the data is located, the Account Class.**

2. **Protect private by avoiding excessive** accessor methods (aka getters) and mutator methods (aka setters). *No mutators allowed in this course.* Do not use a mutator like ~~setBalance~~() which allows any change to the data. The right way to change account balance is to have deposit() and withdrawal().

3. Keep **method names short and simple**.

Methods (like functions) of a class

What makes a class different from a structure is that **the methods** that act on the data of the class **are part of the class**.

The methods are **written in the class source code file** and their **prototype goes in the header file**.

When we declare a method, the we start with the **name of the CLASS**.

`Lumber::print()` this is a function that prints the data from an object of this class.

Output sample: fir 2" x 3" x 6 feet $4.23 in stock: 14

- Classes should have a `toString()` method which returns the formatted data from one object as a string.
- Sometimes classes have a `formatFile()` method which prints the data to the console for debugging.
- The C++ compiler doesn't get confused **because the method call specifies which object it goes with** which tells the compiler **which class's method** and **which specific object's data** to use.

Implied Parameter is Inside the Method

Any function call that uses an object name has an implied parameter of that object.

When I call `albie.print()`, the implied parameter is the instance of Pet class I called `albie` with his data: Type: Beta fish, age: 1, color: blue and gold, Food: Beta food, clean: when nitrogen is above normal.

- Inside the method, I can use this to access the data of the implied parameter if there is any chance the variable name is unclear.
- This is used when we use inheritance from a superclass or when there is a local variable or parameter with the same name as a data member of the class (best to avoid that).

MVC – Model View Controller Architecture of a Program using a Class

Model or Data class - models a real world object. The class stores the state of that object. No main() in this. Two files, **header** and **source code**.

A **collection** or **container (array, vector)** that holds a set of data objects. Can be a separate class or housed in the Controller Program

Controller program manages the sequence of events in a system.

- This controller program implements the logic of the system using the Model class.
- The controller program can have a container as part of the data.
- All data objects of the model class should be in of the controller program.

class Header File in C++

```
1   // Header file for the lumber class                          file: lumber.hpp
2   #include "tools.hpp"
3   #define MAX_STOCK 5
4
5   class LumberT {          // Type definition for a piece of lumber
6     private:
7       string wood;         // Variety of wood.
8       short int height;    // In inches.
9       short int width;     // In inches.
10      short int length;    // In feet.
11      float price;         // Per board, in dollars.
12      int quantity;        // Number of items in stock.
13
14    public:
15      LumberT( const char* wo, int ht, int wd, int ln, float pr, int qt );
16      LumberT() = default;
17      void read();
18      void print()  const ;
19      void sell( int sold );          // Modifies the object.
20      bool equal( const LumberT board2 ) const ;
21
22      string getWood()  const { return wood; }
23      float getPrice()  const { return price; }
24      int getQuantity() const { return quantity; }
25  };
```

Header File for a Class example - LumberT

Note that the file name is written in the upper-right corner of each file.

The data members are the same as the C version except that we are using a C++ string. This makes input easier.

The function prototypes are inside the class declaration (lines 15..24).

This class has two constructors, one with parameters, one without (lines 15..16).

A default constructor is prototyped and defined on line 16. The keyword = `default` makes it very convenient to initialize everything in the object to 0 bits.

Function notes on LumberT.hpp

Note that the **function names here are shorter** than in the C program. This is possible because the **class name, LumberT, is always part of the context** in which these functions are called.

A class function is **called using the name of a class instance**. That class instance is the object that the function reads into or prints or compares. This will become clearer when main() is discussed.

Lines 22. . . 24 are **accessor functions**. Although there are six data members, there are only three getters.

Getters are simply not needed for many class members.

There are no setters at all. Data is put into objects by using the constructor with parameters or using assignment to copy one entire object into another.

Lumber.cpp

Within a class function, the name of a class member refers to that member of whatever object was used to call the function. That object is called **the implied parameter** and it forms the context in which the function executes.

The **constructor with parameters**. Lines 85-92 define the primary class constructor. Note that a constructor does not have a return type. Each line takes the value of one parameter and stores it in the newly-created class instance.

Another common style is to use the **same name for the parameter** and for the class member. In that case, the assignments would be written like this: `this->wood = wood;` **"this" is a keyword**. It is a **pointer to the implied parameter**, and `this ->` can be used to select a part of the object you are initializing.

```
82  #include "lumber.hpp"                                    // file: lumber.cpp
83
84  // Constructor:  ------------------------------------------------------------
85  LumberT::LumberT( const char* wo, int ht, int wd, int ln, float pr, int qt ) {
86      wood = wo;
87      height = ht;
88      width = wd;
89      length = ln;
90      price = pr;
91      quantity = qt;
92  }
93
94  // ------------------------------------------------------------------
95  // Read directly into the object -- no need for a temporary variable.
96  void LumberT::read() {
97      cout <<" Reading a new item.  Enter the 3 dimensions of a board: ";
98      cin >>height >> width >>length;
99      cout <<" Enter the kind of wood: ";
100     cin >> ws;
101     getline( cin, wood );
102     cout <<" Enter the price: $";
103     cin >>price;
104     cout <<" Enter the quantity in stock: ";
105     cin >>quantity;
106 }
107
108 // ------------------------------------------------------------------
109 void LumberT::sell( int sold ){
110     if (quantity >= sold) {
111         quantity -= sold;
112         cout <<" OK, sold " <<sold <<" " <<wood <<endl;
113         print();
114     }
115     else {
116         cout <<" Error: can't sell " <<sold <<" boards; have only "
117              <<quantity <<"\n";
118     }
119 }
120
121 // ------------------------------------------------------------------
122 void LumberT::print() const {
123     cout <<" " <<wood <<":  "
124          <<height <<" x " << width <<", "<<length  <<" feet long -> "
125          <<quantity <<" in stock at $"
126          <<fixed <<setprecision(2) <<price <<endl;
127 }
128
129 // ------------------------------------------------------------------
130 bool LumberT::equal(  const LumberT board2 ) const {
131     return (wood     == board2.wood )  &&
132            (height   == board2.height) &&
133            (width    == board2.width)  &&
134            (length   == board2.length) &&
135            (price    == board2.price)  &&
136            (quantity == board2.quantity);
137  }
```

Lumber.cpp

The read() function. This is very much easier to read and write than the scanf() statements in the C code: no format codes, no ampersands, no fuss. Also, there is no local variable and no return statement.

The data that is read is stored in the implied parameter and is returned that way. Note how easy it is to read input into a C++ string (lines 100-101). There is no need to worry about buffer overflow because strings automatically resize themselves, if needed. The >>ws on line 44 will eliminate the newline character on the end of the previous line plus any whitespace before the visible characters on the current line.

The sell() function is parallel to the C version. Only the output statements have been changed. Notice that print_lumber(*board) in C is simplfied to print() in C++. We don't need a parameter because the object used to call sell() replaces it. We don't need to write anything in front of the function name, because it is called from another function in the same class.

The print() function is parallel to the C version, although the code looks quite different. In C++, class members can be used directly by any class function. So we write height 14 instead of using a parameter name, as in b.height.

The equal() function is exactly like C version except for board1. written in front of each member name. The role played by board1 is covered by the implied parameter in C++

```
82  #include "lumber.hpp"                                    // file: lumber.cpp
83
84  // Constructor:  ---------------------------------------------------------
85  LumberT::LumberT( const char* wo, int ht, int wd, int ln, float pr, int qt ) {
86      wood = wo;
87      height = ht;
88      width = wd;
89      length = ln;
90      price = pr;
91      quantity = qt;
92  }
93
94  // ---------------------------------------------------------------------
95  // Read directly into the object -- no need for a temporary variable.
96  void LumberT::read() {
97      cout <<" Reading a new item.  Enter the 3 dimensions of a board: ";
98      cin >>height >> width >>length;
99      cout <<" Enter the kind of wood: ";
100     cin >> ws;
101     getline( cin, wood );
102     cout <<" Enter the price: $";
103     cin >>price;
104     cout <<" Enter the quantity in stock: ";
105     cin >>quantity;
106 }
107
108 // ---------------------------------------------------------------------
109 void LumberT::sell( int sold ){
110     if (quantity >= sold) {
111         quantity -= sold;
112         cout <<" OK, sold " <<sold <<" " <<wood <<endl;
113         print();
114     }
115     else {
116         cout <<" Error: can't sell " <<sold <<" boards; have only "
117              <<quantity <<"\n";
118     }
119 }
120
121 // ---------------------------------------------------------------------
122 void LumberT::print() const {
123     cout <<" " <<wood <<":  "
124          <<height <<" x " << width <<", "<<length  <<" feet long -> "
125          <<quantity <<" in stock at $"
126          <<fixed <<setprecision(2) <<price <<endl;
127 }
128
129 // ---------------------------------------------------------------------
130 bool LumberT::equal(  const LumberT board2 ) const {
131     return (wood     == board2.wood )  &&
132            (height   == board2.height) &&
133            (width    == board2.width)  &&
134            (length   == board2.length) &&
135            (price    == board2.price)  &&
136            (quantity == board2.quantity);
137  }
```

Using Your Class in C++

You should have **a separate Controller Program that USES the model class** to do something. This is a **separate .cpp source code file**.

- The **controller program** implements the logic of your system and uses the model class to instantiate data objects (create them) and act on the data.
- You can call it `Main.cpp` or name it after your project. *All the programs you have written so far were in a controller program*.
- **The main() function is in this program file** and this where your logic begins.
- You must **include the Class header file in your controller program**, e.g. **`#include "LumberT.hpp"`**
  - **Note that the syntax is different for including a local header file, there are NO ANGLE BRACKETS, we use quotes instead.**
  - This is how C++ knows this is a local class, not a language library.

Include the Class Header to Use the Class

To be able **to use your model class** in your controller program, **you must include the class**.

- The syntax is **a little different** form a library (like `iostream`) **because it is a local header**, so we **use quotes not angle brackets** and the **whole file name of the header file.**

  `#include "Pets.hpp"` //You can use .h or .hpp to name your header, I prefer .hpp

- You **do not include the source code file for the class**, the system knows that there can be a YourClass.cpp when you include YourClass.hpp.
  - The compiler will will look in the same folder where the header was located for the .cpp (e.g. src folder or project folder) for the associated .cpp file.

Object Declarations in the Controller Program

- You can create a **single object of the class or an array of objects, you can use the default data or pass the data to the constructor**.
- To create a **single object of the class**, you can use the following syntax

```
Pets empty; //instance of Pets with default data
Pets blankie ={}; //also an instance of Pets with default data
Pets blackie ={"Blackie", "September", 2011}; //An instance of class with your data
Pets stripes("Stripes", "March", 2020); //Also instantiated with your data
```

An example of method calls with objects

I am using my Pets class and Lumber class in the same program (Job Lot or Walmart)

There are several objects of this class: Albie my beloved Beta, Bit and Byte his neon friends, Julie the dog, Milo the cat, and Nibbles the rabbit.

The class has a `print()` function as well as `feed()`, `clean()`, etc.

- To feed Albie, you call `albie.feed()`, to print his data, call `albie.print()`.
- If I also had Lumber data in this program and I had an object fir2by4 and fir1by8. I could `fir2by4.sell(2)` or `fir2by4.print()`
- **My program would be able to tell the difference** between printing for a Pet and printing for Lumber objects because the call is for one and only one instance of the class (object).

Main.cpp program

Line 30 includes the header for the tools module.

Line 31 includes the header file for the lumber module. You must do this to be able to declare lumber objects and call the lumber functions.

Line 32 is the prototype for the only function that does not belong in the lumber class.

Lines 39-48 create the same lumber objects as in the C program.

Lines 50-52 are exactly the same as the C version.

Compare lines 55-56 to the print_lumber() function in C. C++ output is quite different from C.

- fixed means approximately the same thing as %f.
- To print an amount in dollars and cents, Use fixed and setprecision(2).

```
27  // ------------------------------------------------------------------------
28  // A C++ program corresponding to the Lumber Demo program in C.    main.cpp
29  // ------------------------------------------------------------------------
30  #include "tools.hpp"
31  #include "lumber.hpp"
32  void printInventory ( LumberT stock[], int n, LumberT& onSale );
33
34  // ------------------------------------------------------------------------
35  int main( void ) {
36      cout <<"\n Demo program for structure operations in C++.\n";
37
38      // Struct object declarations: ---------------------------------------
39      LumberT onSale;     // Initialize using the default constructor.
40      LumberT* p;         // Uninitialized
41      LumberT plank( "white oak", 1, 6, 8, 5.80, 158 );  // Call first constr.
42      LumberT stock[5] = { // Call first constructor three times.
43                           { "spruce", 2, 4, 12, 8.20, 27},
44                           { "pine",   2, 3, 10, 5.45, 11},
45                           { "pine",   2, 3, 8, 4.20, 35 },
46                           {} // Call default constructor.
47                           // Call default constructor for additional elements.
48                         };
49
50      int n = 3;                    // The number of items in the stock array.
51      stock[n++] = plank;                              // Copy into the array.
52      printInventory( stock, n, onSale );              // Nothing is on sale.
53
54      // Access parts of a structure ---------------------------------------
55      cout <<fixed <<setprecision(2) <<" "                    // Use object part.
56           <<plank.getWood() <<" is $" <<plank.getPrice() <<"\n ";
57      p = &plank;                                             // Point to object.
58      cout <<plank.getWood() <<" in stock: " <<p->getQuantity() <<endl;
59      p->sell( 200 );                                         // Use the pointer.
60
61      // Use a struct object to call a struct member function. --------------
62      p = &stock[n];              // Point to an object.
63      p->read();                  // Read into stock[4].
64      p->print();                 // Echo-print stock[4].
65
66      printInventory( stock, 5, *p );   // stock[3] is on sale.
67      p->sell( 200 );                   // This will modify the sale object.
68
69      // Test if two structures are equal; Section 13.3.4. -----------
70      printInventory( stock, MAX_STOCK, onSale );
71  }
72
73  // ------------------------------------------------------------------
74  // This function is global, not part of the LumberT structure.
75  void printInventory ( LumberT stock[], int n, LumberT& onSale ){
76      cout <<"\n Our inventory of wood products:\n";
77
78      for (int k = 0; k < n; ++k) {     // Process every slot of array.
79          if ( onSale.equal( stock[k]) )  cout <<" On sale: ";
80          else cout <<"          ";
81          stock[k].print();
82      }
83      cout <<endl;
84  }
```

Main.cpp program

Many people believe that C provides easier ways to control formatting, especially for types float and double.

Lines 63 calls a class function using p, a pointer to a class object. In the C++ program, the object

name is rst, and the function reads info directly into the object p points at. This is a different mode of

operation from the C program, which reads the data into a local temporary and returns a copy of that

temporary that is fnally stored in the object p points at. The C++ program is more direct and more

effcient.

On lines 64 and 67, again, the pointer p provides the context in which the print() and sell() functions

operate, and the functions called are the ones dened for p's object. In the C program, p is written as a

parameter to the print function and the sell function.

```
27  // ------------------------------------------------------------------------
28  // A C++ program corresponding to the Lumber Demo program in C.    main.cpp
29  // ------------------------------------------------------------------------
30  #include "tools.hpp"
31  #include "lumber.hpp"
32  void printInventory ( LumberT stock[], int n, LumberT& onSale );
33
34  // ------------------------------------------------------------------------
35  int main( void ) {
36      cout <<"\n Demo program for structure operations in C++.\n";
37
38      // Struct object declarations: ---------------------------------------
39      LumberT onSale;     // Initialize using the default constructor.
40      LumberT* p;         // Uninitialized
41      LumberT plank( "white oak", 1, 6, 8, 5.80, 158 );  // Call first constr.
42      LumberT stock[5] = { // Call first constructor three times.
43                           { "spruce", 2, 4, 12, 8.20, 27},
44                           { "pine",   2, 3, 10, 5.45, 11},
45                           { "pine",   2, 3, 8, 4.20, 35 },
46                           {} // Call default constructor.
47                           // Call default constructor for additional elements.
48                         };
49
50      int n = 3;                    // The number of items in the stock array.
51      stock[n++] = plank;                              // Copy into the array.
52      printInventory( stock, n, onSale );              // Nothing is on sale.
53
54      // Access parts of a structure ---------------------------------------
55      cout <<fixed <<setprecision(2) <<" "                    // Use object part.
56           <<plank.getWood() <<" is $" <<plank.getPrice() <<"\n ";
57      p = &plank;                                             // Point to object.
58      cout <<plank.getWood() <<" in stock: " <<p->getQuantity() <<endl;
59      p->sell( 200 );                                         // Use the pointer.
60
61      // Use a struct object to call a struct member function. --------------
62      p = &stock[n];              // Point to an object.
63      p->read();                  // Read into stock[4].
64      p->print();                 // Echo-print stock[4].
65
66      printInventory( stock, 5, *p );   // stock[3] is on sale.
67      p->sell( 200 );                   // This will modify the sale object.
68
69      // Test if two structures are equal; Section 13.3.4. -----------
70      printInventory( stock, MAX_STOCK, onSale );
71  }
72
73  // ------------------------------------------------------------------
74  // This function is global, not part of the LumberT structure.
75  void printInventory ( LumberT stock[], int n, LumberT& onSale ){
76      cout <<"\n Our inventory of wood products:\n";
77
78      for (int k = 0; k < n; ++k) {     // Process every slot of array.
79          if ( onSale.equal( stock[k]) )  cout <<" On sale: ";
80          else cout <<"          ";
81          stock[k].print();
82      }
83      cout <<endl;
84  }
```

Wrap - up

Addressing Course Objective 4: Use pointers, strings, arrays, structures, and classes appropriately in programs.

1. Review Structures in C and align with Objects in C++.
2. Be able to define, evaluate appropriateness of use and use an class in C++.
3. Be able to combine class objects with each other and with arrays to represent arbitrarily complex hierarchical objects.
4. Be able to define, evaluate appropriateness of use and use the basic operations on objects .
5. Be able to design classes that meet the design principles of classes.

Resources

Reading chapter 13.4 to end.

Work on programs.