Ch 13: Introduction to Classes CS 2308 Fall 2013 Jill Seaman

13.1 Procedural Programming

- Data is stored in variables
 - Perhaps using arrays and structs.
- Program is a collection of functions that perform operations over the variables
 - Good example: PA2 inventory program
- Variables are passed to the functions as arguments
- Focus is on organizing and implementing the **functions**.

2

Procedural Programming: Problem

- It is not uncommon for
 - program specifications to change
 - representations of data to be changed for internal improvements.
- As procedural programs become larger and more complex, it is difficult to make changes.
 - A change to a given variable or data structure requires changes to all of the functions operating over that variable or data structure.
- Example: use vectors or linked lists instead of arrays for the inventory

Object Oriented Programming: Solution

- An object contains
 - data (like fields of a struct)
 - functions that operate over that data
- Code outside the object can access the data **only** through the object's functions.
- If the representation of the data inside the object needs to change:
 - Only the object's function definitions must be redefined to adapt to the changes.
 - The code outside the object does not need to change, it accesses the object in the same way.

Object Oriented Programming: Concepts

- Encapsulation: combining data and code into a single object.
- **Data hiding** (or **Information hiding**) is the ability to hide the details of data representation from the code outside of the object.
- **Interface**: the mechanism that code outside the object uses to interact with the object.
 - The object's (public) functions
 - Specifically, outside code needs to "know" only the function prototypes (not the function bodies).

Object Oriented Programming: Real World Example

- In order to drive a car, you need to understand only its interface:
 - ignition switch
 - gas pedal, brake pedal
 - steering wheel
 - gear shifter
- You don't need to understand how the steering works internally.
- You can operate any car with the same interface.

Classes and Objects

- A class is like a blueprint for an object.
 - a detailed description of an object.
 - used to make many objects.
 - these objects are called instances of the class.
- For example, the String class in C++.
 - Make an instance (or two):

String cityName1="Austin", cityName2="Dallas";

- use the object's functions to work with the objects:

int size = cityName1.length(); cityName2.insert(0,"Big ");

13.2 The Class

- A class in C++ is similar to a structure.
 - It allows you to define a new (composite) data type.
- A class contains:
 - variables AND
 - functions
- These are called members
- Members can be:
 - private: inaccessible outside the class
 - public: accessible outside the class.

Example class declaration

class Time	//new data type
s stabb iime	// new data cype
private:	
int hour;	
int minute	
void addHou	•
void additor	di (),
public:	
void setHou	ur(int):
void setMin	
int getHour	
	ute() const;
string dis	play() const;
void addMin	
1.	

Access rules

- Used to control access to members of the class
- <u>public</u>: can be accessed by functions inside AND outside of the class
- <u>private</u>: can be called by or accessed only from functions that are members of the class (inside)
 - member variables (attributes) are declared private, to hide their definitions from outside the class.
 - certain functions are declared public to provide (controlled) access to the hidden/private data.
 - these public functions form the interface to the class



9

11

 const appearing after the parentheses in a member function declaration specifies that the function will **not** change any data inside the object.

```
int getHour() const;
int getMinute() const;
string display() const;
```

• These member functions won't change hour or minute.

Defining member functions

- Member function definitions usually occur outside of the class definition (in a separate file).
- The name of each function is preceded by the class name and scope resolution operator (::)

vo	id Time::setHour(int hr)	{
	hour = hr;	
}		
	hour appears to be undefined, but it is a member variable of the Time class	

Accessors and mutators

- Accessor functions
 - return a value from the object (without changing it)
 - a "getter" returns the value of a member variable
- Mutator functions
 - Change the value(s) of member variable(s).
 - a "setter" changes (sets) the value of a member variable.

13

Defining Member Functions

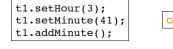
```
void Time::setHour(int hr) {
                       // hour is a member var
 hour = hr:
ł
void Time::setMinute(int min) {
 minute = min;
                       // minute is a member var
}
int Time::getHour() const {
 return hour;
int Time::getMinute() const {
 return minute;
}
void Time::addHour() { // a private member func
 if (hour == 12)
    hour = 1;
 else
    hour++;
```

Defining Member Functions

```
void Time::addMinute() {
  if (minute == 59) {
     minute = 0;
     addHour();
                  // call to private member func
 } else
     minute++;
string Time::display() const {
// returns time in a string formatted to hh:mm
   ostringstream sout; //include <sstream>
   sout.fill('0');
                         //padding char for setw
   sout << hour << ":" << setw(2) << minute;</pre>
   return sout.str(): //str returns the string
                          // from the stream
       ostringstream: allows you to "output" to a string
       using << and i/o manipulators.
                                                    15
       fill(ch): sets padding character used with setw
```

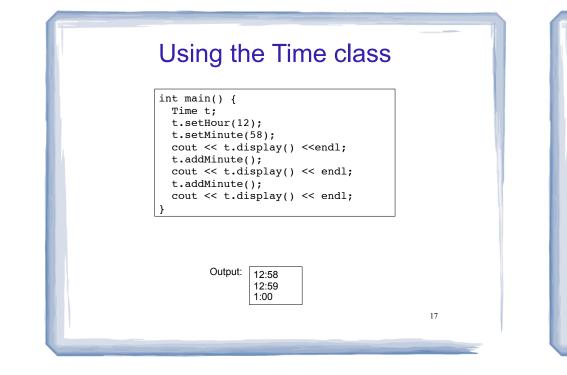
13.3 Defining an instance of the class

- ClassName variable; (like a structure):
- This defines t1 to contain an object of type Time (with hour and minute members).
- Access public members of class with dot notation:





• Use dot notation OUTSIDE the class definitions only.



13.4 Setters and getters: what's the point?

- Why have setters and getters that only do assignment and return values?
 - Why not just make the member variables public?
- Setter functions can validate the incoming data.
 - setMinute can make sure minutes are between 0 and 59 (if not, it can report an error).
- Getter functions could act as a gatekeeper to the data or provide type conversion.

18

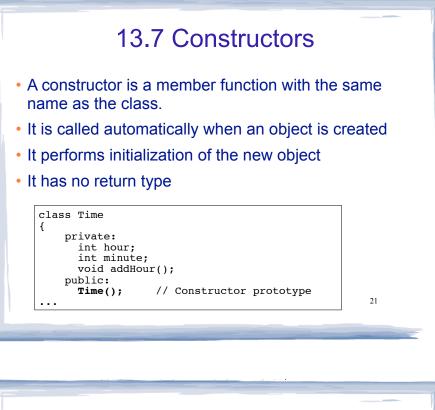
13.5 Separating Specs from Implementation

- Class declarations are usually stored in their own header files (Time.h)
 - called the specification file
- filename is usually same as class name.
- Member function definitions are stored in a separate file (Time.cpp)
 - called the class implementation file
 - it must #include the header file,
- Any program/file using the class must include the class's header file (#include "Time.h") ¹⁹

13.6 Inline member functions

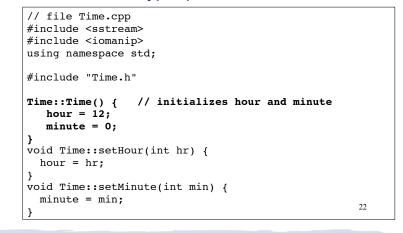
- Member functions can be defined
 - after the class declaration (normally) OR
 - inline: in class declaration
- Inline is appropriate for short function bodies:

<pre>class Time { private: int hour; int minute;</pre>		
<pre>public: int getHour() const { return hour; } int getMinute() const { return minute; } void setHour(int); void setMinute(int);</pre>	<pre>// class decl cont.</pre>	20



Constructor Definition

Note no return type, prefixed with Class::



Constructor "call"

• From main:

<pre>//using Time class (Driver.cpp) #include<iostream> #include "time.h" using namespace std;</iostream></pre>	
<pre>int main() { Time t; //Constructor called implicitly</pre>	here
<pre>cout << t.display() <<endl; t.addMinute(); cout << t.display() << endl; return 0; }</endl; </pre>	
Output: 12:00 12:01	23

Default Constructors

- A default constructor is a constructor that takes no arguments (like Time()).
- If you write a class with NO constructors, the compiler will include a default constructor for you, one that does (almost) nothing.
- The original version of the Time class did not define a constructor, so the compiler provided a constructor for it.

13.8 Passing Arguments to Constructors	Passing Arguments to Constructor
o create a constructor that takes arguments: Indicate parameters in prototype:	Then pass arguments to the constructor when you create an object:
<pre>class Time { public: Time(int,int); // Constructor prototype Use parameters in the definition:</pre>	<pre>int main() { Time t (12, 59); cout << t.display() <<endl; pre="" }<=""></endl;></pre>
<pre>Time::Time(int hr, int min) { hour = hr; minute = min; }</pre>	Output: 12:59
25	26

Classes with no Default Constructor

- When all of a class's constructors require arguments, then the class has NO default constructor.
 - C++ will NOT automatically generate a constructor with no arguments unless your class has NO constructors at all.
- When there are constructors, but no default constructor, you **must** pass the required arguments to the constructor when creating an object.

13.9 Destructors

- Member function that is automatically called when an object is destroyed
- Destructor name is ~classname, e.g., ~Time
- Has no return type; takes no arguments
- Only one destructor per class, i.e., it cannot be overloaded, cannot take arguments
- If the class dynamically allocates memory, the destructor should release (delete) it

Destructors	
Example: class decl Inventory class, with dynamically allocated array	y:
<pre>struct Product { string productName; // product description string locator; // used to find product int quantity; // number of copies in inve double price; // selling price of the pro };</pre>	
<pre>class Inventory { private: Product *products; //dynamically allocated arr int count; public: Inventory (int); ~Inventory(); bool addItem(Product); int removeItem(String); //name of Product to r void showInventory(); } }</pre>	-
<pre>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>></pre>	29

Destructors

• Example: member function definitions (class impl)

#include "Inventory.h"	Inventory.cpp
<pre>Inventory::Inventory(int size){ products = new Product[size]; count = 0; }</pre>	
<pre>Inventory::~Inventory() { delete [] products; }</pre>	
	30

Destructors

· Example: driver creates and destroys an Inventory

int main() {

Inventory storeProducts(100); //calls constructor

//do stuff with storeProducts here

return 0;

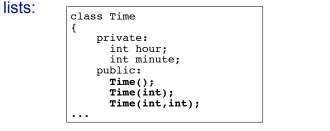
} //end of prog, storeProducts object destroyed here, // calls its destructor (deletes products array)

- When is an object destroyed?
 - at the end of its scope
 - when it is deleted (if it's dynamically allocated)

31

13.10 Overloaded Constructors

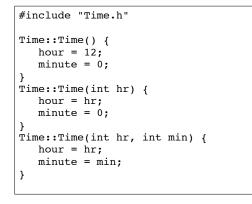
- Recall: when 2 or more functions have the same name they are *overloaded*.
- A class can have more than one constructor
- They have the same name, so they are overloaded
- Overloaded functions must have different parameter



32



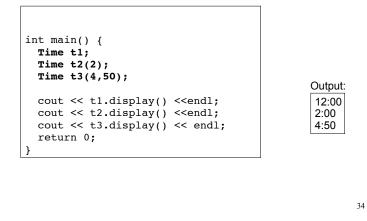
• definitions:



33

Overloaded Constructor "call"

• From main:



Overloaded Member Functions

- Non-constructor member functions can also be overloaded
- Must have unique parameter lists as for constructors

```
class Time
{
    private:
        int hour;
        int minute;
    public:
        Time();
        Time(int);
        Time(int,int);
        void addMinute(); //adds one minute
        void addMinute(int); //adds minutes from arg
...
35
```

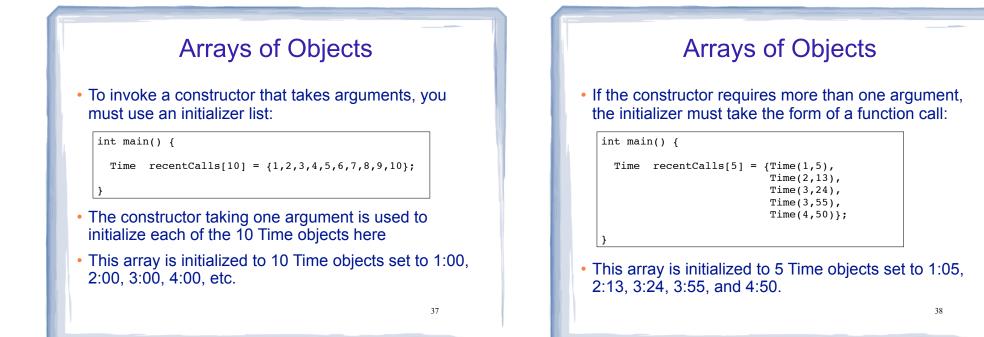
13.12 Arrays of Objects

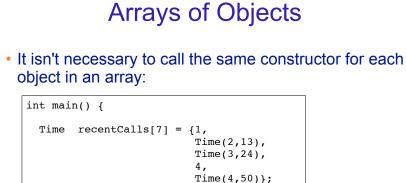
• Objects can be the elements of an array:

int main() {

Time recentCalls[10]; //times of last 10 calls

- Default constructor (Time()) is used to initialize each element of the array when it is defined
- This array is initialized to 10 Time objects each set to 12:00.





• If there are fewer initializers in the list than elements in the array, the default constructor will be called for all the remaining elements.

39

Accessing Objects in an Array

- Objects in an array are referenced using subscripts
- Member functions are referenced using dot notation
- Must access the specific object in the array BEFORE calling the member function:

recentCalls[2].setMinute(30); cout << recentCalls[4].display() << endl;</pre>