#### Introduction to Classes 13.2 The Class A class in C++ is similar to a structure. Unit 4 - It allows you to define a new (composite) data type. A class contains the following: Chapter 13 variables AND - functions (these manipulate the variables) CS 2308 These are called members Spring 2020 A class declaration defines the member variables and the prototypes of the member **Jill Seaman** functions. 2

3

## Example class declaration

// models a 1	2 hour clock
class Time {	//new data type
private:	
int hour;	
int minut	e;
void addH	
public:	
void setH	our(int);
void setM	linute(int);
int getHo	our() const;
int getMi	<pre>nute() const;</pre>
2	
string di	<pre>splay() const;</pre>
void addM	
}:	

Access specifiers

- Used to control access to members of the class
  - <u>public</u> members can be accessed by functions inside AND outside of the class
  - <u>private</u> members can be called or accessed only from functions inside the class (the class's member functions) Private is the default setting for class members.
- Member variables 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

## Using const with member functions

 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.
- Other functions may or may not change them.
- using const here is optional.

## Accessors and mutators

- Accessor functions
  - return a value from the object (without changing it)
  - can be defined using const.
  - a "getter" is a special accessor function that returns the value of **one** member variable
- Mutator functions
  - Change the value(s) of member variable(s).
  - a "setter" is a special mutator function that changes (sets) the value of **one** member variable.

## **Defining member functions**

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

```
void Time::setHour(int hr) {
    hour = hr;
}
hour appears to be undefined.
```

but it is a member variable of the Time class

7

## **Defining Member Functions**

```
void Time::setHour(int hr) {
                       // hour is a member var
 hour = hr;
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 string formatted to hh:mm
    string hourString = to string(hour);
    string minuteString = to string(minute);
    if (minuteString.length()==1)
        minuteString = "0" + minuteString;
    return hourString + ":" + minuteString;
to string(int): converts an int to string.
```

string.length(): returns number of chars in string. str1+str2: returns a new string formed by adding chars of str1 followed by chars of str2.

## 13.3 Defining an instance of the class

- ClassName variable; Time t1;
- This defines t1 to contain an object of type Time (with hour and minute members).
- Then access public members of class with dot notation.

t1.setHour(3); t1.setMinute(41); t1.addMinute();

calls to member functions

10

like declaring a structure variable

# Using the Time class

```
int main() {
 Time t;
  t.setHour(12);
  t.setMinute(58);
 cout << t.display() <<endl;</pre>
  t.addMinute();
 cout << t.display() << endl;</pre>
 t.addMinute();
  cout << t.display() << endl;</pre>
```

Output: 12:58 12:59 1:00

Note: the program includes the code from slides 3, 8, 9, and 11 (and any #includes needed). See AllTime.cpp in timedemo.zip

# **13.1 Procedural Programming**

- A style of programming in which:
- Data is stored in variables
  - Perhaps using arrays and structs.
- Program is a collection of functions that perform operations over the variables
  - Good example: Programming Assignment 2
- Variables are passed to the functions as arguments
- Focus is on organizing and implementing the functions. 12

9

# 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 list of lab access events

# Object Oriented Programming: Solution

- An object (instance of a class) 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:

17

int size = cityName1.length();

cityName2.append(" Cowboys");

# 13.5 Separating Specs from Implementation

- Class declarations are usually stored in their own "header files" (Time.h)
  - called the specification file
- Member function definitions are stored in a separate file (Time.cpp)
  - called the class implementation file
- Main function and standalone functions go in a third file (Driver.cpp)



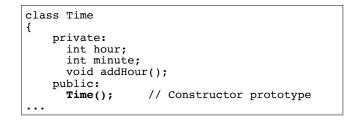
# 13.6 Inline member functions

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

```
class Time {
  private:
    int hour;
    int minute;
    void addHour(); // not inlined
  public:
                            return hour; }
    int getHour() const {
    int getMinute() const {
                            return minute; }
    void setHour(int h) {
                            hour = h; }
    void setMinute(int m) {
                            minute = m; }
    string display() const; //not inlined
    void addMinute();
                             //not inlined
                                                  19
};
```

# 13.7 Constructors

- A constructor is a member function with the same name as the class.
- · It is called automatically when an object is created
- It performs initialization of the new object
- It has no return type

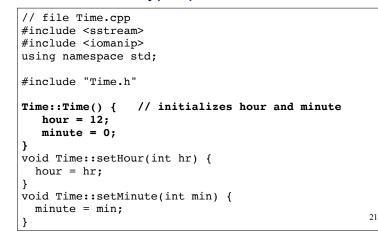


20

18

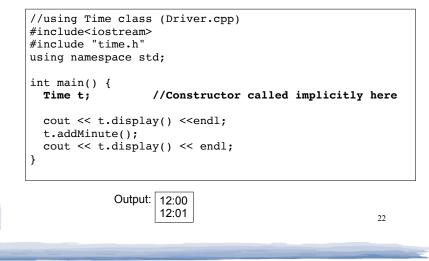
## **Constructor Definition**

• Note no return type, prefixed with Class::



# Constructor "call"

#### From main:



# 13.8 Passing Arguments to Constructors

- To create a constructor that takes arguments:
  - Indicate the parameters in the prototype:

```
class Time
{
    public:
        Time(int,int); // Constructor prototype
...
```

23

- Use the parameters in the definition:

```
Time::Time(int hr, int min) {
    hour = hr;
    minute = min;
}
```

# **Passing Arguments to Constructors**

• Pass arguments to the constructor when you create an object (in the declaration):

int main() {				
Time t (12, 59);				
<pre>cout &lt;&lt; t.display() &lt;<endl;< pre=""></endl;<></pre>				
1				



## **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.

#### 25

## **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.

#### 26

## 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 (it cannot be overloaded, cannot take arguments).
- If the class dynamically allocates memory, the destructor should release (delete) it

## Destructors

• Example: Inventory class, with a dynamically allocated array of part numbers:

# Inventory { private: String \*parts; //dynamically allocated array int count; public: Inventory (int); ~Inventory(); //destructor bool addPart(string); int removePart(string); void showInventory(); };

## Destructors

• Example: member function definitions for constructor and destructor:

#include "Inventory.h"	Inventory.cpp
<pre>Inventory::Inventory(int size){    parts = new String[size]; //dynamic all    count = 0;</pre>	ocation
}	
<pre>Inventory::~Inventory() {     delete [] parts; }</pre>	

29

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 function
  - They have the same name, so they are overloaded
- Overloaded functions must have different parameter

lists:		
11515.	class Time	
	{	
	private:	
	int hour;	
	int minute;	
	public:	
	Time();	
	Time(int);	
	Time(int, int);	
	•••	

### Destructors

#### Example: driver creates and destroys an Inventory

int main() {

Inventory inv(100); //calls constructor, allocates array

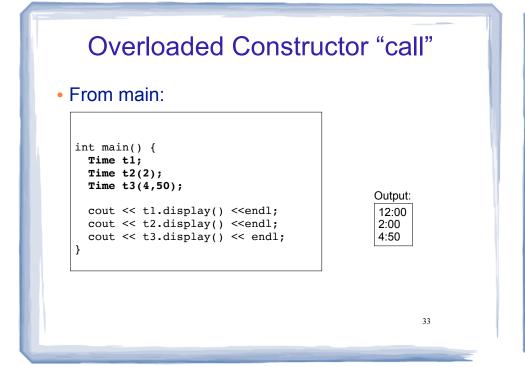
- //do stuff with inv here
- } //end of main, inv object destroyed here, // calls its destructor (which deletes parts array)
- When is an object destroyed?
  - at the end of its scope (regular variables) OR
  - when it is deleted (if it's dynamically allocated)

## **Overloaded Constructors**

#### • definitions:

```
#include "Time.h"
Time::Time() {
    hour = 12;
    minute = 0;
}
Time::Time(int hr) {
    hour = hr;
    minute = 0;
}
Time::Time(int hr, int min) {
    hour = hr;
    minute = min;
}
```

32



# 13.12 Arrays of Objects

 An array can contain objects (the element type can be a Class):

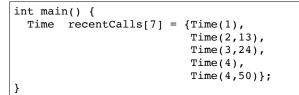
int main() {

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

- The default constructor (Time()) is used to initialize each element of the array when it is created.
- This array is initialized to 10 Time objects, each set to 12:00.
- To invoke a constructor that takes arguments, you must use an initializer list . . . 34

Arrays of Objects initializer lists

• Each initializer takes the form of a function call:



- If there are fewer initializers in the list than elements in the array, the default constructor will be called for all the remaining elements.
- This array is initialized to 7 Time objects, set to 1:00, 2:13, 3:24, 4:00, 4:50, 12:00 and 12:00. 35

# 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>
```

• Processing array elements in a loop:

```
for (int i=0; i<7; i++)
    cout << recentCalls[i].display() << " ";
cout << endl;</pre>
```

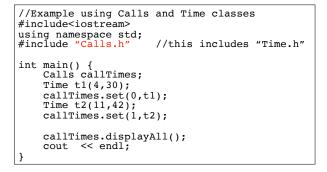
# Composition

#### • When one class contains another as a member:

	Calls.h
phone calls ng default constr	uctor
	Calls.cpp
//calls member	function
	ng default constr

# Composition

#### • Driver for Calls



#### Output:

4:30 11:42 12:00 12:00 12:00 12:00 12:00 12:00 12:00 12:00 38