Lecture Notes

Chapter 10

Classes and Data Abstraction

ECE 111: Introduction to C and C++ Programming

Instructor: Dr. Shayan (Sean) Taheri

Gannon University (GU)

Personal Information

- Name: Shayan (Sean) Taheri.
- Date of Birth: July/28/1991.
- Current Position: Assistant Professor at Gannon University
- Previous Position: Postdoctoral Fellow at University of Florida.
- Ph.D. Degree: Electrical Engineering from the University of Central Florida.
- M.S. Degree: Computer Engineering from the Utah State University.
- University Profile: https://www.gannon.edu/FacultyProfiles.aspx?profile=taheri001

Objectives (1 of 2)

In this chapter, you will:

- Learn about classes
- Learn about **`private`**, **`protected`**, and **`public`** members of a class
- Explore how classes are implemented
- Become aware of accessor and mutator functions
- Examine constructors and destructors

Objectives (2 of 2)

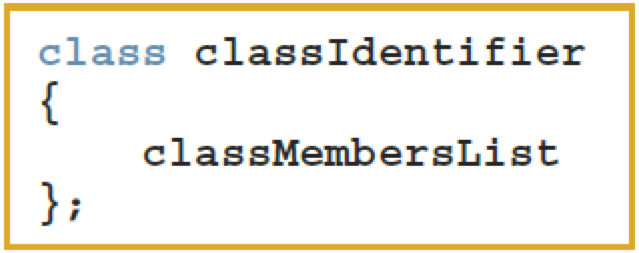
- Learn about the abstract data type (ADT)
- Explore how classes are used to implement ADTs
- Become aware of the differences between a `struct` and a `class`
- Learn about information hiding
- Explore how information hiding is implemented in C++
- Become aware of inline functions of a class
- Learn about the `static` members of a class

Classes (1 of 4)

- Object-oriented design (OOD): a problem solving methodology
- Object: combines data and the operations on that data in a single unit
- Class: a collection of a fixed number of components
- Member: a component of a class

Classes (2 of 4)

- The general syntax for defining a **`class`**:



```
class classIdentifier
{
     classMembersList
};
```

- A class member can be a variable or a function
- If a member of a **`class`** is a variable
  - It is declared like any other variable
  - You cannot initialize a variable when you declare it

Classes (3 of 4)

- If a member of a `class` is a function
  - A function prototype declares that member
  - Function members can (directly) access any member of the **`class`**
- A class definition defines only a data type
  - No memory is allocated
  - Remember the semicolon (`;`) after the closing brace

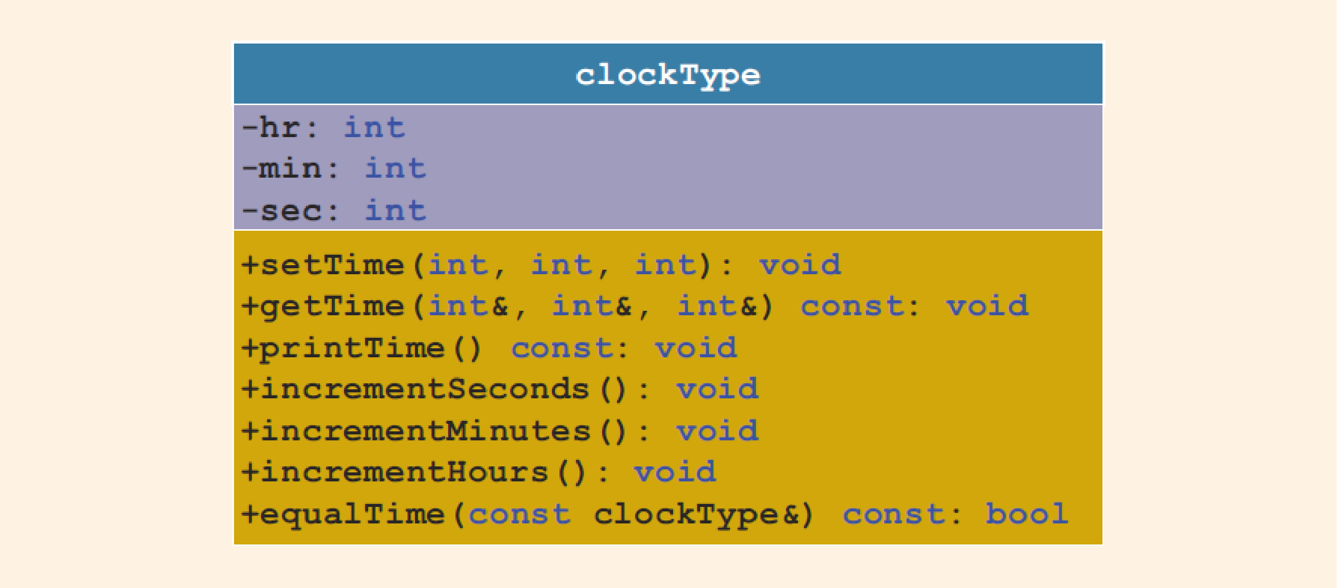
Classes (4 of 4)

- Three categories of class members:
  - `private` (default)
    - Member cannot be accessed outside the `class`
  - `public`
    - Member is accessible outside the class
  - `protected`

Unified Modeling Language Class Diagrams (1 of 2)

- Unified Modeling Language (UML) notation: used to graphically describe a class and its members
  - **+**: member is public
  - **−**: member is private
  - **#**: member is protected

Unified Modeling Language Class Diagrams (2 of 2)



```
clockType
-hr: int
-min: int
-sec: int
+setTime(int, int, int): void
+getTime(int&, int&, int&) const: void
+printTime() const: void
+incrementSeconds(): void
+incrementMinutes(): void
+incrementHours(): void
+equalTime(const clockType&) const: bool
```

FIGURE 10-1 UML class diagram of the `class clockType`

Variable (Object) Declaration

- Once defined, you can declare variables of that `class` type
  - `clockType myClock;`
  - `clockType yourClock;`
- A `class` variable is called a class object or class instance

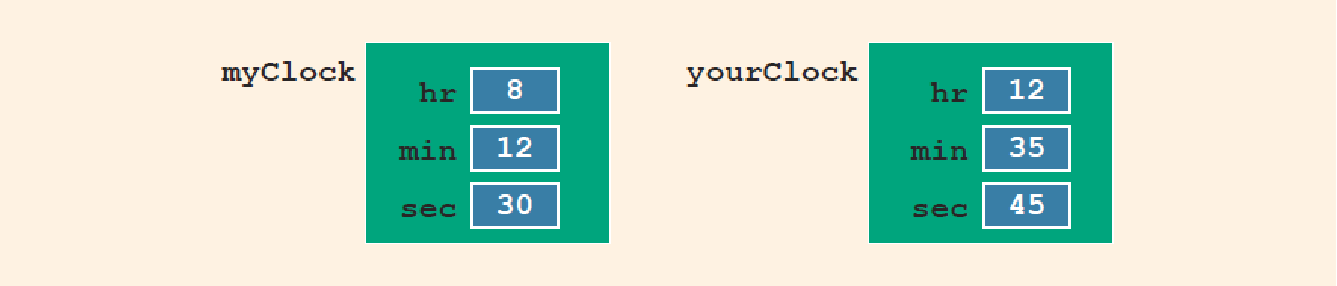


FIGURE 10-2 Objects `myClock` and `yourClock`

Accessing Class Members

- Once an object is declared, it can access the members of the class
- The general syntax for an object to access a member of a class:

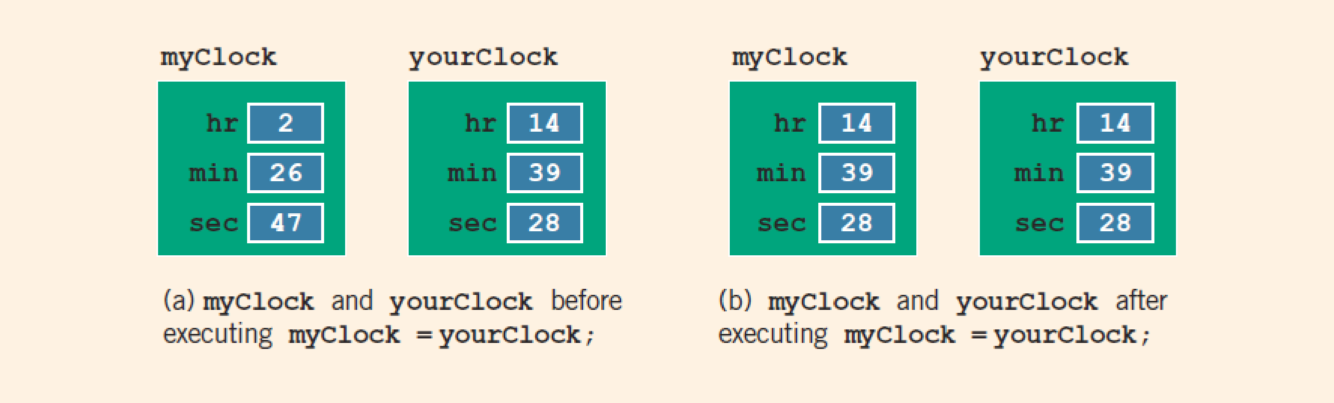
```
classObjectName.memberName
```

- If an object is declared in the definition of a member function of the class, it can access the `public` and `private` members
- The dot (`.`) is the member access operator

Built-in Operations on Classes

- Most of C++'s built-in operations do not apply to classes
  - Arithmetic operators cannot be used on class objects unless the operators are overloaded
  - Relational operators cannot be used to compare two class objects for equality
- Built-in operations that are valid for class objects:
  - Member access (**.**)
  - Assignment (**=**)

Assignment Operator and Classes



(a) **myClock** and **yourClock** before executing **myClock = yourClock;**

(b) **myClock** and **yourClock** after executing **myClock = yourClock;**

FIGURE 10-3 `myClock` and `yourClock` before and after executing the statement `myClock = yourClock;`

Class Scope (1 of 2)

- A **class** object can be automatic or static
  - Automatic: created when the declaration is reached and destroyed when the surrounding block is exited
  - Static: created when the declaration is reached and destroyed when the program terminates
- A member of a `class` has the same scope as a member of a `struct`

Class Scope (2 of 2)

- A member of the `class` is local to the `class`
- You access a `class` member outside the `class` by using the `class` object name and the member access operator (`.`)

Functions and Classes

- Objects can be passed as parameters to functions and returned as function values
- As parameters to functions:
  - Class objects can be passed by value or by reference
- If an object is passed by value:
  - Contents of data members of the actual parameter are copied into the corresponding data members of the formal parameter

Reference Parameters and Class Objects (Variables) (1 of 2)

- Passing by value might require a large amount of storage space and a considerable amount of computer time to copy the value of the actual parameter into the formal parameter
- If a variable is passed by reference:
  - The formal parameter receives only the address of the actual parameter

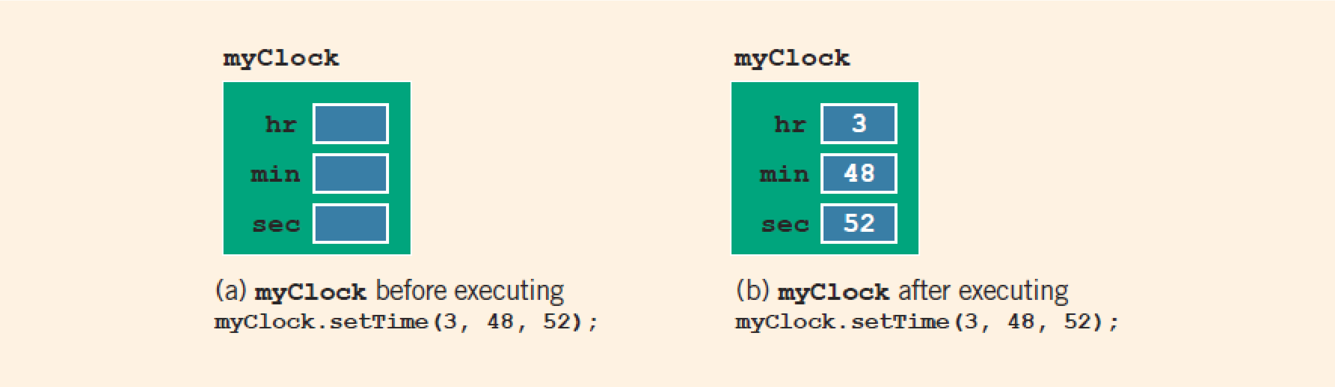
Reference Parameters and Class Objects (Variables) (2 of 2)

- Pass by reference is an efficient way to pass a variable as a parameter
  - Problem: when passing by reference, the actual parameter changes when the formal parameter changes
  - Solution: use `const` in the formal parameter declaration

Implementation of Member Functions (1 of 4)

- Must write the code for functions defined as function prototypes
- Prototypes are left in the class to keep the class smaller and to hide the implementation
- To access identifiers local to the class, use the scope resolution operator, (: :)

Implementation of Member Functions (2 of 4)



(a) **`myClock`** before executing `myClock.setTime(3, 48, 52);`

(b) **`myClock`** after executing `myClock.setTime(3, 48, 52);`

FIGURE 10-4 **`myClock`** before and after executing the statement **`myClock.setTime(3, 48, 52);`**

Implementation of Member Functions (3 of 4)

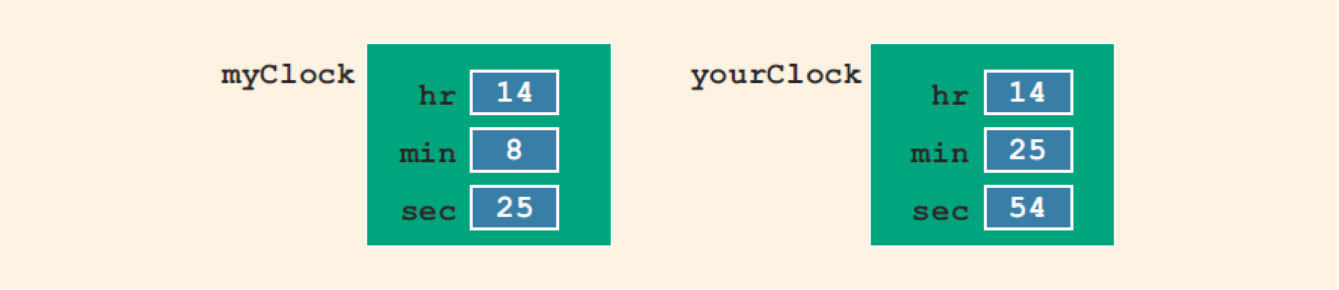


FIGURE 10-5 Objects **`myClock`** and **`yourClock`**

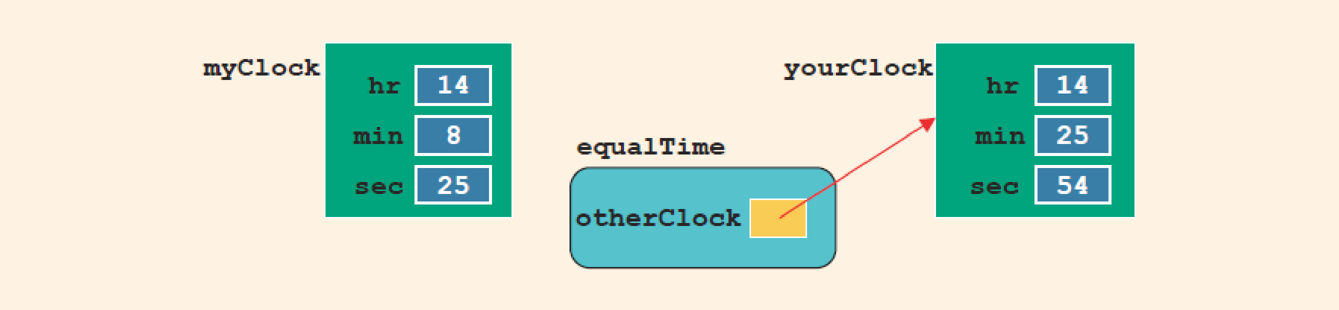


FIGURE 10-6 Object **`myClock`** and parameter **`otherClock`**

Implementation of Member Functions (4 of 4)

- Once a class is properly defined and implemented, it can be used in a program
  - A program that uses/manipulates objects of a class is called a client of that class
- When you declare objects of the `class clockType`, each object has its own copy of the member variables (**`hr`**, **`min`**, and **`sec`**)
  - These variables are called instance variables of the class
  - Every object has its own copy of the data

Accessor and Mutator Functions

- Accessor function: member function that only accesses the value(s) of member variable(s)
- Mutator function: member function that modifies the value(s) of member variable(s)
- Constant member function
  - Member function that cannot modify member variables of that class
  - Member function heading with `const` at the end

Order of `public` and `private` Members of a Class

- C++ has no fixed order in which to declare **`public`** and **`private`** members
- By default, all members of a class are **`private`**
- Use the member access specifier **`public`** to make a member available for **`public`** access

Constructors (1 of 2)

- Use constructors to guarantee that member variables of a class are initialized
- Two types of constructors
  - With parameters
  - Without parameters (default constructor)
- Other properties of constructors
  - Name of a constructor is the same as the name of the class
  - A constructor has no type

Constructors (2 of 2)

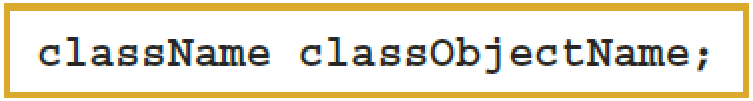
- A class can have more than one constructor
  - Each must have a different formal parameter list
- Constructors execute automatically when a class object enters its scope
  - They cannot be called like other functions
- Which constructor executes depends on the types of values passed to the class object when the class object is declared

Invoking a Constructor

- A constructor is automatically executed when a class variable is declared
- Because a class may have more than one constructor, you can invoke a specific constructor

Invoking the Default Constructor

- Syntax to invoke the default constructor is:



```
className classObjectName;
```

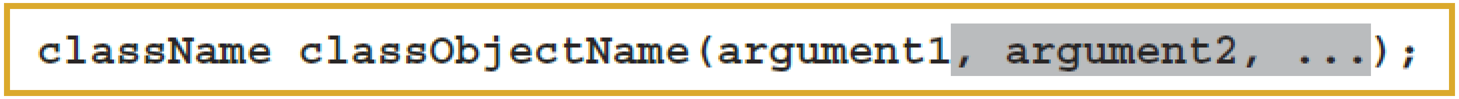
- The statement:

```
clockType yourClock;
```

declares **`yourClock`** to be an object of type **`clockType`** and the default constructor executes

Invoking a Constructor with Parameters

- The syntax to invoke a constructor with a parameter is:



```
className classObjectName(argument1, argument2, ...);
```

- Number and type of arguments should match the formal parameters (in the order given) of one of the constructors
  - Otherwise, C++ uses type conversion and looks for the best match
  - Any ambiguity causes a compile-time error

Constructors and Default Parameters

- A constructor can have default parameters
  - Rules for declaring formal parameters are the same as for declaring default formal parameters in a function
  - Actual parameters are passed according to the same rules for functions
- A default constructor is a constructor with no parameters or with all default parameters

Classes and Constructors: A Precaution

- If a class has no constructor(s), C++ provides the default constructor
  - However, the object declared is still uninitialized
- If a class includes constructor(s) with parameter(s), but not the default constructor
  - C++ does not provide the default constructor
  - Appropriate arguments must be included when the object is declared

In-line Initialization of Data Members and the Default Constructor

- C++11 standard allows member initialization in class declarations
  - Called in-line initialization of the data members
- When an object is declared without parameters, then the object is initialized with the in-line initialized values
  - If declared with parameters, then the default values are overridden by the constructor with the parameters

Arrays of Class Objects (Variables) and Constructors

- If you declare an array of class objects, the class should have the default constructor
  - The default constructor is typically used to initialize each (array) class object

Destructors

- Destructors are functions without any type
- A class can have only one destructor
  - The destructor has no parameters
- The name of a destructor is the tilde character (**~**) followed by the class name
  - Example: **`~clockType();`**
- The destructor automatically executes when the class object goes out of scope

Data Abstract, Classes, and Abstract Data Types

- Abstraction
  - Separating design details from usage
  - Separating the logical properties from the implementation details
- Abstraction also applicable to data
- Abstract data type (ADT): a data type that separates the logical properties from the implementation details
- Three things associated with an ADT
  - Type name: the name of the ADT
  - Domain: the set of values belonging to the ADT
  - Set of operations on the data

A struct versus a class (1 of 2)

- By default, members of a **struct** are **public**
  - **private** specifier can be used in a **struct** to make a member private
- By default, the members of a **class** are **private**
- **class**es and **struct**s have the same capabilities

A `struct` versus a `class` (2 of 2)

- In C++, the definition of a **`struct`** was expanded to include member functions, constructors, and destructors
- If all member variables of a **`class`** are **`public`** and there are no member functions:
  - Use a **`struct`**

Information Hiding (1 of 3)

- Information hiding refers to hiding the details of the operations on the data
- The header file (or interface file) contains the specification details
  - The header file has an extension **`h`**
- The implementation file contains the definitions of the functions to implement the operations of an object
  - This file has an extension **`cpp`**
- In the header file, include function prototypes and comments that briefly describe the functions
  - Specify preconditions and/or postconditions

Information Hiding (2 of 3)

- Implementation file must include the header file via the **`include`** statement
- In the **`include`** statement:
  - User-defined header files are enclosed in double quotes
  - System-provided header files are enclosed between angular brackets

Information Hiding (3 of 3)

- Precondition: a statement specifying the condition(s) that must be true before the function is called
- Postcondition: a statement specifying what is true after the function call is completed

Executable Code

- To use an object in a program
  - The program must be able to access the implementation details of the object
- IDEs Visual C++ Express (2013 or 2016) and Visual Studio 2015, and C++ Builder put the editor, compiler, and linker into a package
  - One command (build, rebuild, or make) compiles program and links it with the other necessary files
  - These systems also manage multiple file programs in the form of a project

More Examples of Classes

- Various examples of classes and how to use them in a program are presented
- Refer to Example 10-8 through Example 10-11

Inline Functions

- An inline function definition is a member function definition given completely in the definition of the class
  - Saves the overhead of a function invocation
- Very short definitions should be defined as inline functions

`static` Members of a Class (1 of 2)

- Use the keyword **`static`** to declare a function or variable of a class as **`static`**
- A **`public static`** function or member of a class can be accessed using the class name and the scope resolution operator
- **`static`** member variables of a class exist even if no object of that **`class`** type exists

static Members of a Class (2 of 2)

- Multiple objects of a class each have their own copy of non-**static** member variables
- All objects of a class share any **static** member of the class

Quick Review (1 of 3)

- A `class` is a collection of a fixed number of components
- Components of a `class` are called the members of the `class`
  - Accessed by name
  - Classified into one of three categories: `private`, `protected`, and `public`
- In C++, `class` variables are called `class` objects or `class` instances or, simply, objects

Quick Review (2 of 3)

- The only built-in operations on classes are assignment and member selection
- Constructors guarantee that data members are initialized when an object is declared
    - A default constructor has no parameters
- The destructor automatically executes when a class object goes out of scope
    - A `class` can have only one destructor
    - The destructor has no parameters

Quick Review (3 of 3)

- An abstract data type (ADT) is a data type that separates the logical properties from the implementation details
- A `public static` member, function or data, of a `class` can be accessed using the `class` name and the scope resolution operator, `::`
- `static` member variables of a `class` exist even when no object of the `class` type exists
- Instance variables are non-`static` data members

Reading Assignment – Very Important for “GU – ECE 111”

- Malik, D.S., 2014. **C++ programming: Program design including data structures**. Cengage Learning.
  - “**Chapter 10**: **Classes and Data Abstraction**”.