Classes

CPE 212

Outline

- Quick Review of C++ **structs**
- Introduction to C++ Classes
- Compiling Multi-file Programs
- Include Guards

Quick Review of C++ structs

Records (C++ structs)

- ***Record*** (***structure*** in C++) – a ***structured data type*** with a fixed number of components that are accessed by name. The components may be heterogeneous (of different types)
- ***Field*** (***member*** in C++) – a component of a record

struct Declaration Example

```
struct  StudentRec
{
   string    firstName;
   string    lastName;
   float     gpa;
};
```

TypeName

MemberList

Important Observation:
StudentRec is the name of a ***new data type*** created by the Programmer. The declaration above *does not allocate memory*. You must declare a variable of the new type to allocate memory.

struct Variable Declaration Example

StudentRec **nextStudent;**

Data Type

Variable Identifier

Important Observation:
The syntax for declaring a struct variable is the <u>same</u> as that for declaring a variable of a Simple data type.

struct Member Access Examples

```
nextStudent.firstName = "Homer";
nextStudent.gpa = 3.0;

cout << nextStudent.firstName << endl;
```

Struct Variable Name

Member Selection Operator

Member Name

Introduction to C++ Classes

Class Concepts - 1

- ***Abstact Data Type (ADT)***
  - A data type whose properties (domain and operations) are specified independently of any implementation

Class Concepts - 2

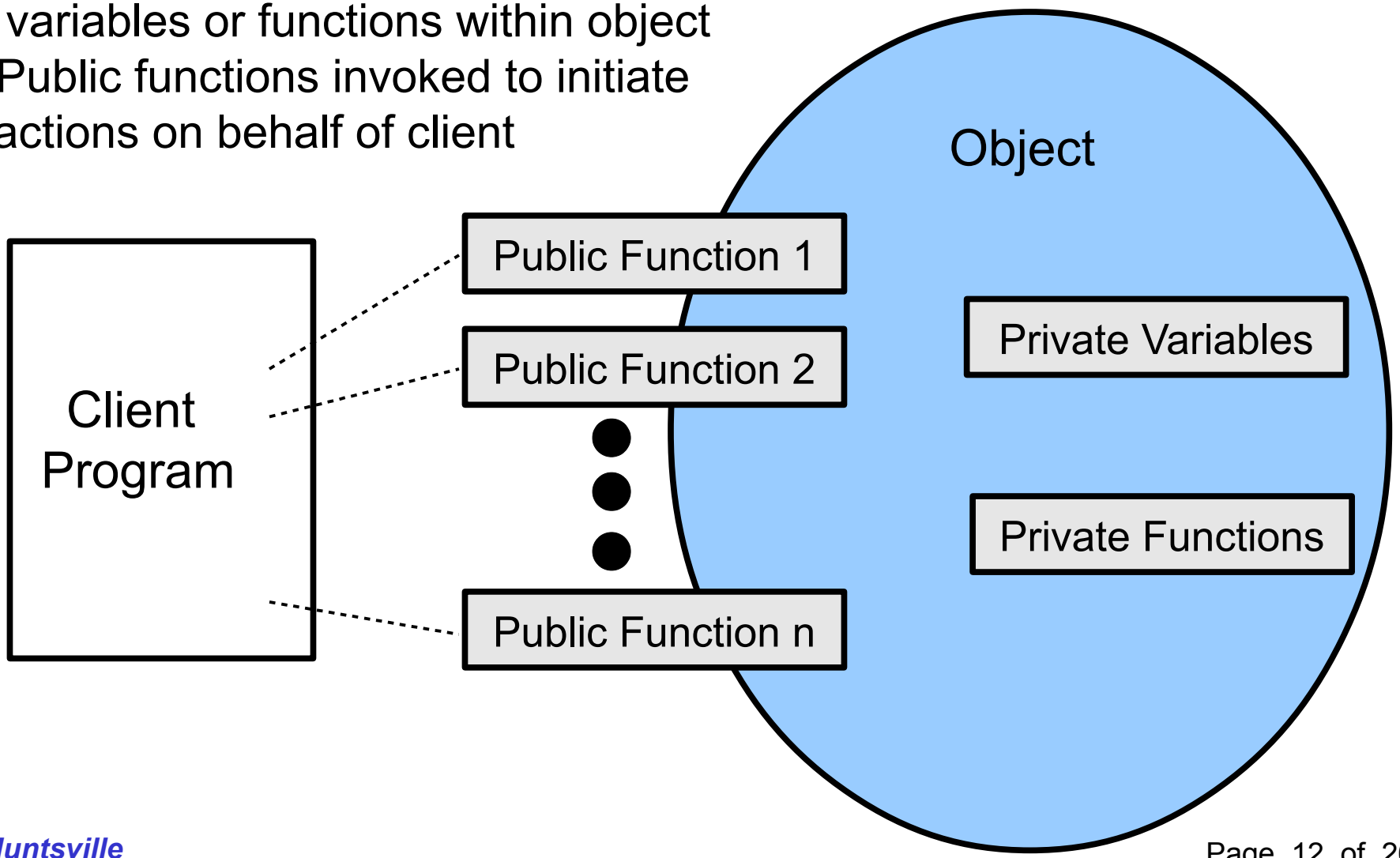
- ***Class***
  - A structured type used to model abstract data types
  - Encapsulate attributes (data) with the member functions that modify the attribute values
    - Examples of Classes in C++: string, ifstream, ofstream
- ***Object***
  - An instance of a class
  - Set of attribute values define the state of an object at a given time
  - Member functions and attributes accessed using the ***member selection operator*** (period .)
- ***Client***
  - Software that declares and manipulates objects of a particular class type

Class Concepts - 3

- Accessibility of class members
  - A **public** member
    - may be directly accessed from outside the class
  - A **private** member
    - accessible only by the code within the implementation file and is not accessible by code outside of the class
  - We will discuss **protected** members later

Class Concepts - 4

- Client cannot interact directly with private variables or functions within object
- Public functions invoked to initiate actions on behalf of client



Class Concepts - 5

- Classes are typically written in two parts
  - ***Specification File*** (classname.h)
    - The declaration of the class type
    - Include guards (more on this later)
  - ***Implementation File*** (classname.cpp)
    - Code that implements the member functions of the class
- **Unit testing** of a class is performed with a dedicated driver program (classdriver.cpp)
  - Contains a simplified main function that creates instances of the class (objects) and then tests the objects by using its public interface
  - Multiple source files must be compiled and linked to create the executable
  - Once tested, the class may be reused with the actual application program

Class Concepts - 6

```
…
struct StudentRec      // Type Declaration
{
    string   lastName;
    string   firstName;
    float    gpa;
};
int main()
{
  StudentRec  someStudent, nextStudent;  // Variable Declarations
  …
  nextStudent = someStudent;   // Member by member copy using =
  someStudent.gpa = 4.0;       // Access via member selector op (.)
  …
}
```

- Comparison of a **struct** and a **class**
  - A **struct** is a **class** whose members are **public** by default
  - By default, all members of a C++ class are **private**
  - **Two built-in operations** for **structs** and **classes** **.** and **=**

Class Concepts - 7

- Major categories of member functions
  - **Constructors**
    - Create and initialize objects
  - **Transformers**
    - Alter the state of an object
  - **Observers**
    - Allow one to view the state of an object
  - **Iterators**
    - Allow us to process, one at a time, all components of an ADT
  - **Destructors**
    - Allow us to clean up when an object is no longer needed

Class Concepts - 8

- const Member Functions
  - Member functions applied to an object may alter attributes stored within that object unless the reserved word const is used to prevent modification
- self
  - The object to which a member function is applied

Time Class Example - 1

- **Declare a class to represent the Time ADT**
  - Time in HH:MM:SS

- **Where do we begin?**
  - Identify the key attributes and their default values
  - Identify the key operations to perform
  - Write the class declaration (*.h)
  - Define the member functions (*.cpp)
  - Write a simple client driver program to test the Time class

Time Class Example - 2

- **What are the key attributes?**
  - **Hours**: valid range 0 through 23 inclusive
  - **Minutes**: valid range 0 through 59 inclusive
  - **Seconds**: valid range 0 through 59 inclusive
- **What are reasonable default values for these attributes?**
  - Hours = 0
  - Minutes = 0
  - Seconds = 0

Time Class Header File

```
//************************** time.h ******************************
// Homer Simpson, Project XYZ, CPE 212-01
// Purpose: Declaration of class to represent Time ADT
//
// Note:  On an actual project, you would be provided with a header file such as this one.
//        You would NOT submit this file for grading!!
//*****************************************************************
class Time
{
  private:    // Private members here
    int hrs;                                        // Valid range 0-23 inclusive
    int mins;                                       // Valid range 0-59 inclusive
    int secs;                                       // Valid range 0-59 inclusive

  protected: // Protected members here -- none required

  public:     // Public members here
    /******* Constructors *******/
    Time();                                         // Default constructor sets Time to 0:0:0

    Time(int initHrs, int initMins, int initSecs);   // Parameterized constructor: Constructs Time using incoming parameters

    /******* Transformers *******/
    void Set(int hours, int minutes, int seconds ); // Sets Time based on incoming parameters

    void Increment();                               // Time has been advanced by one second,
                                                    // with 23:59:59 wrapping around to 0:0:0

    /******* Observers *******/
    void Write() const;                             // Time has been output in the form HH:MM:SS

    bool Equal(Time otherTime ) const;              // Function value == true, if this time equals otherTime;
                                                     // value false otherwise

    bool LessThan(Time otherTime ) const;           // Function value == true, if this time is earlier;
                                                     // value false otherwise
                                                     // Assuming this time and otherTime represent times in the same day

    /******* Iterators and Destructors - none *******/
};
```

Constructor methods have the same name as the class

Time Class Implementation File - 1

```
//********* time.cpp Standard CPE212 Implementation Header Here ********
//
// Note:  On an actual project, you write and submit an implementation file such as this one
//
#include <iostream>

#include "time.h"        // Preprocessor directive which inserts the contents of the
                         // specified file at this location prior to compile

using namespace std;

Time::Time()  // Default constructor - Sets hrs == 0  &&  mins == 0  &&  secs == 0
{
  hrs = 0;
  mins = 0;
  secs = 0;
} // End Default Constructor


Time::Time(int initHrs, int initMins, int initSecs)         // Parameterized Constructor
// Makes hrs == initHrs  &&  mins == initMins  &&  secs == initSecs
// Assumes values are in allowable range
{
  hrs = initHrs;
  mins = initMins;
  secs = initSecs;
} // End Parameterized Constructor
```

Time Class Implementation File - 2

```
void Time::Set(int hours, int minutes, int seconds)   // Set
// Sets hrs == hours  &&  mins == minutes  &&  secs == seconds assuming values in range
{
  hrs = hours;
  mins = minutes;
  secs = seconds;
} // End Time::Set(…)


void Time::Increment()   // Increment
// Advances time by one second, with 23:59:59 wrapping around to 0:0:0
{
    secs++;
    if (secs > 59)
    {
        secs = 0;
        mins++;
        if (mins > 59)
        {
            mins = 0;
            hrs++;
            if (hrs > 23)
                hrs = 0;
        }
    }
} // End Time::Increment()
```

Time Class Implementation File - 3

```
void Time::Write()  const  // Write()
//     Time has been output in the form HH:MM:SS
{
    if (hrs < 10)
        cout << '0';
    cout << hrs << ':';
    if (mins < 10)
        cout << '0';
    cout << mins << ':';
    if (secs < 10)
        cout << '0';
    cout << secs;
} // End Time::Write()

bool Time::Equal( Time otherTime ) const   // Equal
//     Function value == true, if this time equals otherTime; value == false otherwise
//                    == false, otherwise
{
    return (hrs == otherTime.hrs && mins == otherTime.mins && secs == otherTime.secs);
} // End Time::Equal(…)


bool Time::LessThan( Time otherTime ) const   // LessThan
// Assume this time and otherTime represent times in the same day
//     Function value == true, if this time is earlier in the day than otherTime; value == false otherwise
{
    return (hrs < otherTime.hrs ||
            hrs == otherTime.hrs && mins < otherTime.mins ||
            hrs == otherTime.hrs && mins == otherTime.mins
                                 && secs < otherTime.secs);
} // End LessThan(…)
```

Time Driver File

```
//*****************************************************************************
// timedriver.cpp program
// The purpose of the driver is to unit test the time class
//
//
//*****************************************************************************
#include <iostream>

#include "time.h"

using namespace std;

int main()
{









    return 0;
}  // End timedriver
```

Compiling Multi-File Programs - 1

- The hard way…
  - Generate an object file (.o) from each .cpp file

```
g++   -c  time.cpp
g++   -c  timedriver.cpp
```

  - Link the .o files to create the executable file

```
g++   time.o  timedriver.o   -o timedriver
```

- Works but cumbersome -- especially when you have many files to compile and link

Compiling Multi-File Programs - 2

- ***make*** utility
  - Write a **makefile** that describes how to compile and link your files and save this file with your source files
  - Don't forget the **TAB**
  - Type **make** at the prompt to rebuild your program after any changes to any of the source files
  - See the **make** tutorial on Angel for more details

```
timedriver:  timedriver.o  time.o
     g++   time.o  timedriver.o   -o timedriver

time.o:  time.h  time.cpp
     g++   -c  time.cpp

timedriver.o:  time.h  timedriver.cpp
     g++   -c  timedriver.cpp
```

Include Guards

```
//************************** time.h ******************************
// Homer Simpson, Project XYZ, CPE 212-01
// Purpose: Declaration of class to represent Time ADT
//****************************************************************
#ifndef TIME_H
#define TIME_H

class Time
{
  private:     // Private members here
    int hrs;                                          // Valid range 0-23 inclusive
    int mins;                                         // Valid range 0-59 inclusive
    int secs;                                         // Valid range 0-59 inclusive

  protected:   // Protected members here -- none required

  public:      // Public members here
    /******* Constructors *******/
    Time();                                           // Default constructor sets Time to 0:0:0

    Time(int initHrs, int initMinsint initSecs );     // Parameterized constructor: Constructs Time using incoming parameters


    /******* Transformers *******/
    void Set(int hours, int minutes, int seconds );   // Sets Time based on incoming parameters

    void Increment();                                 // Time has been advanced by one second,
                                                      // with 23:59:59 wrapping around to 0:0:0



    /******* Observers *******/
    void Write() const;                               // Time has been output in the form HH:MM:SS

    bool Equal(Time otherTime ) const;                // Function value == true, if this time equals otherTime;
                                                       // value false otherwise

    bool LessThan(Time otherTime ) const;             // Function value == true, if this time is earlier;
                                                       // value false otherwise
                                                     // Assuming this time and otherTime represent times in the same day

    /******* Iterators - none *******/

    /******* Destructors - none *******/
};
#endif
```