CPE 212 - Fundamentals of Software Engineering

•••

Classes

Objective:
Overview of the use of classes

Outline

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

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

- `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.

Type Name

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

Member List

struct Variable Declaration

- The syntax for declaring a struct variable is the same as that for declaring a variable of a Simple data type.

Data Type — Variable Identifer

```
StudentRec nextStudent;
```

struct Member Access

```
nextStudent.firstName = “Homer”;
nextStudent.gpa = 3.0;

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

Struct Variable Name

Member Name

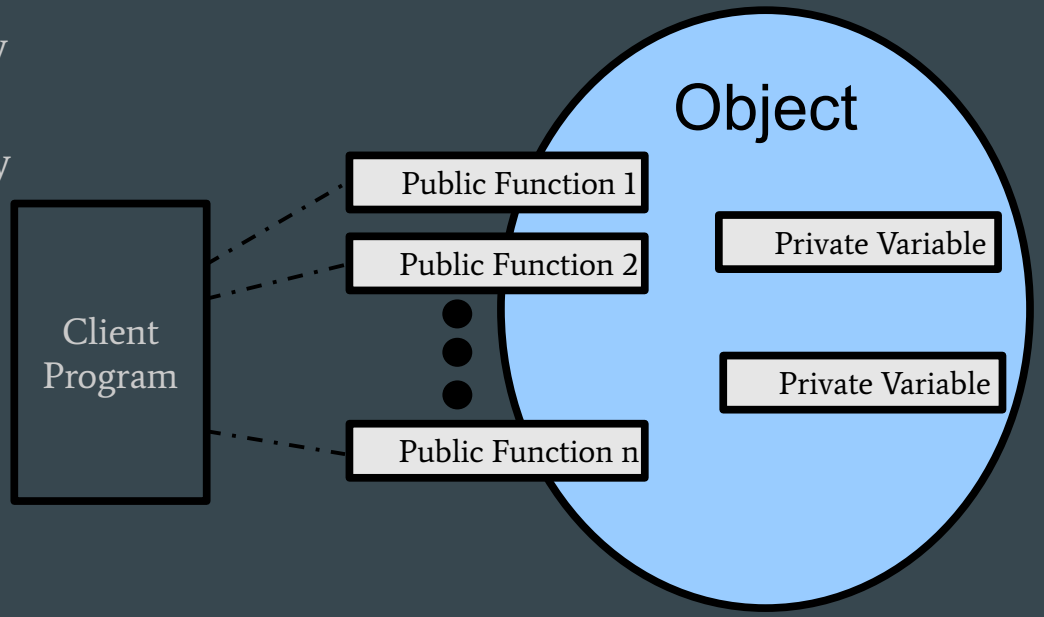
Member Selection Operator

Class Concepts

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

- 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



Abstraction

- ATM Actions
  - Cash Withdrawal
  - Money Transfer
  - Balance Inquiry
- How does it all work?

Class Concepts

- 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

- 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 =

```
...
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 (.)
  ...
}
```

Class Concepts

- 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
- 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

Class Concepts

- 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
- 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

- 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
- 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.h

- Constructor methods have the same name as the class

```
#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 initMins, int initSecs );   // 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;

};
#endif
```

time.cpp

```cpp
//********* 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.cpp

```cpp
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.cpp

```cpp
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(...)
```

Compiling Multi-File Programs

- 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

- 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 Canvas 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
```

Header Guards

- Conditional compilation directives that tells the preprocessor to check to see if the unique identifier has already been defined
- Prevents duplicate inclusion

```
//************************** 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 initMins, int initSecs );

    /******* Transformers *******/
    void Set(int hours, int minutes, int seconds );
    void Increment();
    /******* Observers *******/
    void Write() const;
    bool Equal(Time otherTime ) const;
    bool LessThan(Time otherTime ) const;
    /******* Iterators - none *******/
    /******* Destructors - none *******/
};
#endif
```