Classes: A Deeper Look

Class Construction: C++

Contents

- Time Class Case Study and preprocessor wrappers
- Accessing Class Members
- Destructors
- Default Memberwise Assignment

Time Class Case Study

Preprocessor wrapper

- using a “preprocessor wrapper” in header files to prevent the code in the header from being included into the same source code file more than once.
- Such a preprocessor directive prevents multiple-definition errors.

```
// Declaration of class Time.
// Member functions are defined in Time.cpp

// prevent multiple inclusions of header file
#ifndef TIME_H
#define TIME_H

// Time class definition
class Time
{
public:
   Time(); // constructor
   void setTime( int, int, int ); // set hour, minute and second
   void printUniversal(); // print time in universal-time format
   void printStandard(); // print time in standard-time format
private:
   int hour; // 0 - 23 (24-hour clock format)
   int minute; // 0 - 59
   int second; // 0 - 59
}; // end class Time

#endif
```

```cpp
#include <iostream>
#include <iomanip>
#include "Time.h" // include definition of class Time from Time.h
using namespace std;

// Time constructor initializes each data member to zero.
// Ensures all Time objects start in a consistent state.
Time::Time()
{
   hour = minute = second = 0;
} // end Time constructor

// set new Time value using universal time; ensure that
// the data remains consistent by setting invalid values to zero
void Time::setTime( int h, int m, int s )
{
   hour = ( h >= 0 && h < 24 ) ? h : 0; // validate hour
   minute = ( m >= 0 && m < 60 ) ? m : 0; // validate minute
   second = ( s >= 0 && s < 60 ) ? s : 0; // validate second
} // end function setTime

// print Time in universal-time format (HH:MM:SS)
void Time::printUniversal()
{
   cout << setfill( '0' ) << setw( 2 ) << hour << ":"
      << setw( 2 ) << minute << ":" << setw( 2 ) << second;
} // end function printUniversal

// print Time in standard-time format (HH:MM:SS AM or PM)
void Time::printStandard()
{
   cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 ) << ":"
      << setfill( '0' ) << setw( 2 ) << minute << ":" << setw( 2 )
      << second << ( hour < 12 ? " AM" : " PM" );
} // end function printStandard
```

```cpp
// Program to test class Time.
// NOTE: This file must be compiled with Time.cpp.
#include <iostream>
#include "Time.h" // include definition of class Time from Time.h
using namespace std;

int main()
{
   Time t; // instantiate object t of class Time

   // output Time object t's initial values
   cout << "The initial universal time is ";
   t.printUniversal(); // 00:00:00
   cout << "\nThe initial standard time is ";
   t.printStandard(); // 12:00:00 AM

   t.setTime( 13, 27, 6 ); // change time

   // output Time object t's new values
   cout << "\n\nUniversal time after setTime is ";
   t.printUniversal(); // 13:27:06
   cout << "\nStandard time after setTime is ";
   t.printStandard(); // 1:27:06 PM

   t.setTime( 99, 99, 99 ); // attempt invalid settings

   // output t's values after specifying invalid values
   cout << "\n\nAfter attempting invalid settings:"
      << "\nUniversal time: ";
   t.printUniversal(); // 00:00:00
   cout << "\nStandard time: ";
   t.printStandard(); // 12:00:00 AM
   cout << endl;
} // end main
```

Time Class: Constructors w. Default Arguments

```
// Time class containing a constructor with default arguments.
// Member functions defined in Time.cpp.

// prevent multiple inclusions of header file
#ifndef TIME_H
#define TIME_H

// Time abstract data type definition
class Time
{
public:
   Time( int = 0, int = 0, int = 0 ); // default constructor

   // set functions
   void setTime( int, int, int ); // set hour, minute, second
   void setHour( int ); // set hour (after validation)
   void setMinute( int ); // set minute (after validation)
   void setSecond( int ); // set second (after validation)
```

Time Class: Constructors w. Default Arguments

```
   // get functions
   int getHour(); // return hour
   int getMinute(); // return minute
   int getSecond(); // return second

   void printUniversal(); // output time in universal-time format
   void printStandard(); // output time in standard-time format
private:
   int hour; // 0 - 23 (24-hour clock format)
   int minute; // 0 - 59
   int second; // 0 - 59
}; // end class Time

#endif
```

Time Class Case Study

```
The initial universal time is 00:00:00
The initial standard time is 12:00:00 AM

Universal time after setTime is 13:27:06
Standard time after setTime is 1:27:06 PM

After attempting invalid settings:
Universal time: 00:00:00
Standard time: 12:00:00 AM
```

- Once class Time has been defined, it can be used as a type in object, array, pointer and reference declarations as follows:

```
Time sunset; // object of type Time
Time arrayOfTimes[ 5 ]; // array of 5 Time objects
Time &dinnerTime = sunset; // reference to a Time object
Time *timePtr = &dinnerTime; // pointer to a Time object
```

Class Scope and Accessing Class Members

- A class's data members and member functions belong to that class's scope.
- Within a class's scope, class members are immediately accessible by all of that class's member functions and can be referenced by name.

- Outside a class's scope: Public class members are referenced through one of the **handles** on an object—an object name, a reference to an object or a pointer to an object. The type of the object, reference or pointer specifies the interface (i.e., the member functions) accessible to the client.

- The dot member selection operator (.) is preceded by an object's name or with a reference to an object to access the object's members.

- The **arrow member selection operator (->)** is preceded by a pointer to an object to access the object's members.

```cpp
// Demonstrating the class member access operators . and ->
#include <iostream>
using namespace std;

// class Count definition
class Count
{
public: // public data is dangerous
   // sets the value of private data member x
   void setX( int value )
   {
      x = value;
   } // end function setX

   // prints the value of private data member x
   void print()
   {
      cout << x << endl;
   } // end function print

private:
   int x;
}; // end class Count
```

Class Members Access Operators . And ->

```
int main()
{
   Count counter; // create counter object
   Count *counterPtr = &counter; // create pointer to counter
   Count &counterRef = counter; // create reference to counter

   cout << "Set x to 1 and print using the object's name: ";
   counter.setX( 1 ); // set data member x to 1
   counter.print(); // call member function print

   cout << "Set x to 2 and print using a reference to an object: ";
   counterRef.setX( 2 ); // set data member x to 2
   counterRef.print(); // call member function print

   cout << "Set x to 3 and print using a pointer to an object: ";
   counterPtr->setX( 3 ); // set data member x to 3
   counterPtr->print(); // call member function print
} // end main
```

```
Set x to 1 and print using the object's name: 1
Set x to 2 and print using a reference to an object: 2
Set x to 3 and print using a pointer to an object: 3
```

NOTE: Utility Function

```
// SalesPerson class definition.
// Member functions defined in SalesPerson.cpp.
#ifndef SALESP_H
#define SALESP_H

class SalesPerson
{
public:
   static const int monthsPerYear = 12; // months in one year
   SalesPerson(); // constructor
   void getSalesFromUser(); // input sales from keyboard
   void setSales( int, double ); // set sales for a specific month
   void printAnnualSales(); // summarize and print sales
private:
   double totalAnnualSales(); // prototype for utility function
   double sales[ monthsPerYear ]; // 12 monthly sales figures
}; // end class SalesPerson

#endif
```

Destructors

- A **destructor** is another type of special member function.
- The name of the destructor for a class is the **tilde character (~)** followed by the class name.

- A class's destructor is called implicitly when an object is destroyed. This occurs, for example, as an automatic object is destroyed when program execution leaves the scope in which that object was instantiated.
- The destructor itself does not actually release the object's memory—it performs **termination housekeeping** before the object's memory is reclaimed, so the memory may be reused to hold new objects.

- A destructor receives no parameters and returns no value.
- A destructor may not specify a return type—not even void.
- A class may have only one destructor—destructor overloading is not allowed.
- A destructor must be public.

When Constructors and Destructors Are Called

- Constructors and destructors are called implicitly by the compiler.
- The order in which these function calls occur depends on the order in which execution enters and leaves the scopes where the objects are instantiated.

- Generally, destructor calls are made in the reverse order of the corresponding constructor calls,
- But, the storage classes of objects can alter the order in which destructors are called.

- Example: Constructors defined in global scope: the corresponding destructors are called when main terminates.

Let us demonstrates the order in which constructors and destructors are called

Constructors and Destructors: Calling Rules

- The constructor for an automatic local object is called when execution reaches the point where that object is defined—the corresponding destructor is called when execution leaves the object's scope
- Constructors and destructors for automatic objects are called each time execution enters and leaves the scope of the object.
- Destructors are not called for automatic objects if the program terminates with a call to function exit or function abort.

- The constructor for a static local object is called only once, when execution first reaches the point where the object is defined—the corresponding destructor is called when main terminates or the program calls function exit.

- Global and static objects are destroyed in the reverse order of their creation. Destructors are not called for static objects if the program terminates with a call to function abort.

```cpp
// CreateAndDestroy class definition.
// Member functions defined in CreateAndDestroy.cpp.
#include <string>
using namespace std;

#ifndef CREATE_H
#define CREATE_H

class CreateAndDestroy
{
public:
   CreateAndDestroy( int, string ); // constructor
   ~CreateAndDestroy(); // destructor
private:
   int objectID; // ID number for object
   string message; // message describing object
}; // end class CreateAndDestroy

#endif
```

```
// CreateAndDestroy class member-function definitions.
#include <iostream>
#include "CreateAndDestroy.h"// include CreateAndDestroy class definition
using namespace std;

// constructor
CreateAndDestroy::CreateAndDestroy( int ID, string messageString )
{
   objectID = ID; // set object's ID number
   message = messageString; // set object's descriptive message

   cout << "Object " << objectID << "   constructor runs   "
      << message << endl;
} // end CreateAndDestroy constructor

// destructor
CreateAndDestroy::~CreateAndDestroy()
{
   // output newline for certain objects; helps readability
   cout << ( objectID == 1 || objectID == 6 ? "\n" : "" );

   cout << "Object " << objectID << "   destructor runs    "
      << message << endl;
} // end ~CreateAndDestroy destructor
```

```cpp
// Demonstrating the order in which constructors and
// destructors are called.
#include <iostream>
#include "CreateAndDestroy.h" // include CreateAndDestroy class definition
using namespace std;

void create( void ); // prototype

CreateAndDestroy first( 1, "(global before main)" ); // global object

int main()
{
   cout << "\nMAIN FUNCTION: EXECUTION BEGINS" << endl;
   CreateAndDestroy second( 2, "(local automatic in main)" );
   static CreateAndDestroy third( 3, "(local static in main)" );

   create(); // call function to create objects

   cout << "\nMAIN FUNCTION: EXECUTION RESUMES" << endl;
   CreateAndDestroy fourth( 4, "(local automatic in main)" );
   cout << "\nMAIN FUNCTION: EXECUTION ENDS" << endl;
} // end main

// function to create objects
void create( void )
{
   cout << "\nCREATE FUNCTION: EXECUTION BEGINS" << endl;
   CreateAndDestroy fifth( 5, "(local automatic in create)" );
   static CreateAndDestroy sixth( 6, "(local static in create)" );
   CreateAndDestroy seventh( 7, "(local automatic in create)" );
   cout << "\nCREATE FUNCTION: EXECUTION ENDS" << endl;
} // end function create
```

Default Memberwise Assignment

- The assignment operator (=) can be used to assign an object to another object of the same type.

- By default, such assignment is performed by memberwise assignment:
  - each data member of the object on the right of the assignment operator is assigned individually to the same data member in the object on the left of the assignment operator.

```
date2 = date1; // default memberwise assignment
```

```
// Date class declaration.  Member functions are defined in Date.cpp.

// prevent multiple inclusions of header file
#ifndef DATE_H
#define DATE_H

// class Date definition
class Date
{
public:
   Date( int = 1, int = 1, int = 2000 ); // default constructor
   void print();
private:
   int month;
   int day;
   int year;
}; // end class Date

#endif
```

```cpp
// Date class member-function definitions.
#include <iostream>
#include "Date.h" // include definition of class Date from Date.h

using namespace std;

// Date constructor (should do range checking)
Date::Date( int m, int d, int y )
{
   month = m;
   day = d;
   year = y;
} // end constructor Date

// print Date in the format mm/dd/yyyy
void Date::print()
{
   cout << month << '/' << day << '/' << year;
} // end function print
```

```cpp
// Demonstrating that class objects can be assigned
// to each other using default memberwise assignment.
#include <iostream>
#include "Date.h" // include definition of class Date from Date.h
using namespace std;

int main()
{
   Date date1( 7, 4, 2004 );
   Date date2; // date2 defaults to 1/1/2000

   cout << "date1 = ";
   date1.print();
   cout << "\ndate2 = ";
   date2.print();

   date2 = date1; // default memberwise assignment

   cout << "\n\nAfter default memberwise assignment, date2 = ";
   date2.print();
   cout << endl;
} // end main
```

```
date1 = 7/4/2004
date2 = 1/1/2000

After default memberwise assignment, date2 = 7/4/2004
```