W6.1

• Destructors
• Data Members and Member Functions
• Returning a Reference to a Private Data Member
• Default Memberwise Copy
• Software Reusability
6.12 Using Destructors

• Destructors
  – Are member function of class
  – Perform termination housekeeping before the system reclaims the object’s memory
  – Complement of the constructor
  – Name is tilde (~) followed by the class name (i.e., ~Time)
    • Recall that the constructor’s name is the class name
  – Receives no parameters, returns no value
  – One destructor per class
    • No overloading allowed
6.13 When Constructors and Destructors Are Called

• Constructors and destructors called automatically
  – Order depends on scope of objects

• Global scope objects
  – Constructors called before any other function (including main)
  – Destructors called when main terminates (or exit function called)
  – Destructors not called if program terminates with abort

• Automatic local objects
  – Constructors called when objects are defined
  – Destructors called when objects leave scope
    • i.e., when the block in which they are defined is exited
  – Destructors not called if the program ends with exit or abort
6.13 When Constructors and Destructors Are Called

• Static local objects
  – Constructors called when execution reaches the point where the objects are defined
  – Destructors called when `main` terminates or the `exit` function is called
  – Destructors not called if the program ends with `abort`
1 // Fig. 6.9: create.h
2 // Definition of class CreateAndDestroy.
3 // Member functions defined in create.cpp.
4 #ifndef CREATE_H
5 #define CREATE_H
6
7 class CreateAndDestroy {
8 public:
9    CreateAndDestroy( int ); // constructor
10    ~CreateAndDestroy();   // destructor
11 private:
12    int data;
13};
14
15 #endif
2. Load the header file

2.1 Modify the constructor and destructor

Constructor and Destructor changed to print when they are called.
33 // Fig. 6.9: fig06_09.cpp
34 // Demonstrating the order in which constructors and
destructors are called.
36 #include <iostream>
37 
38 using std::cout;
39 using std::endl;
40 
41 #include "create.h"
42 
43 void create( void );   // prototype
44 
45 CreateAndDestroy first( 1 );  // global object
46 
47 int main()
48 {
49    cout << "   (global created before main)" << endl;
50
51    CreateAndDestroy second( 2 );        // local object
52    cout << "   (local automatic in main)" << endl;
53
54    static CreateAndDestroy third( 3 );  // local object
55    cout << "   (local static in main)" << endl;
56
57    create();  // call function to create objects
58
59    CreateAndDestroy fourth( 4 );        // local object
60    cout << "   (local automatic in main)" << endl;
61    return 0;
62 }
// Function to create objects
void create( void )
{
    CreateAndDestroy fifth( 5 );
cout << " (local automatic in create)" << endl;
static CreateAndDestroy sixth( 6 );
cout << " (local static in create)" << endl;

    CreateAndDestroy seventh( 7 );
cout << " (local automatic in create)" << endl;
}
6.14 Using Data Members and Member Functions

• Member functions
  – Allow clients of the class to set (i.e., write) or get (i.e., read) the values of private data members
  – Example:
    
    *Adjusting a customer’s bank balance
    
  • `private` data member `balance` of a class `BankAccount` could be modified through the use of member function `computeInterest`
  • A member function that sets data member `interestRate` could be called `setInterestRate`, and a member function that returns the `interestRate` could be called `getInterestRate`
  
  – Providing `set` and `get` functions does not make `private` variables `public`
  
  – A set function should ensure that the new value is valid
6.15 A Subtle Trap: Returning a Reference to a Private Data Member

• Reference to an object
  – Alias for the name of the object,
  – May be used on the left side of an assignment statement, makes perfectly acceptable \textit{lvalue}.
  – Reference can receive a value, which changes the original object as well

• Returning references
  – \texttt{public} member functions can return non-\texttt{const} references to \texttt{private} data members
    • Should be avoided, breaks encapsulation

• Please avoid using references in this way, very, very bad!!!
1 // Fig. 6.11: time4.h
2 // Declaration of the Time class.
3 // Member functions defined in time4.cpp
4
5 // preprocessor directives that
6 // prevent multiple inclusions of header file
7 #ifndef TIME4_H
8 #define TIME4_H
9
10 class Time {
11 public:
12    Time( int = 0, int = 0, int = 0 );
13    void setTime( int, int, int );
14    int getHour();
15    int &badSetHour( int );  // DANGEROUS reference return
16 private:
17    int hour;
18    int minute;
19    int second;
20 };
// Fig. 6.11: time4.cpp
// Member function definitions for Time class.
#include "time4.h"

// Constructor function to initialize private data.
// Calls member function setTime to set variables.
// Default values are 0 (see class definition).
Time::Time( int hr, int min, int sec )
    { setTime( hr, min, sec ); }

// Set the values of hour, minute, and second.
void Time::setTime( int h, int m, int s )
{
    hour   = ( h >= 0 && h < 24 ) ? h : 0;
    minute = ( m >= 0 && m < 60 ) ? m : 0;
    second = ( s >= 0 && s < 60 ) ? s : 0;
}

// Get the hour value
int Time::getHour() { return hour; }

// POOR PROGRAMMING PRACTICE:
// Returning a reference to a private data member.
int &Time::badSetHour( int hh )
{
    hour = ( hh >= 0 && hh < 24 ) ? hh : 0;
    return hour; // DANGEROUS reference return
}
1.2 Declare reference

Declare Time object \( t \) and reference \( \text{hourRef} \) that is assigned the reference returned by the call \( t.\text{badSetHour}(20) \).

2. Change data using a reference

Alias used to set the value of \text{hour} to 30 (an invalid value).

3. Output results

Function call used as an \textit{lvalue} and assigned the value 74 (another invalid value).

POOR PROGRAMMING PRACTICE!!!!!!!!!
badSetHour as an lvalue, Hour: 74

*********************************
POOR PROGRAMMING PRACTICE!!!!!!!!!

*********************************
HourRef used to change hour to an invalid value. Normally, the function setbadSetHour would not have allowed this. However, because it returned a reference, hour was changed directly.

Hour before modification: 20
Hour after modification: 30

*********************************
POOR PROGRAMMING PRACTICE!!!!!!!
badSetHour as an lvalue, Hour: 74
*********************************
6.16 Assignment by Default Memberwise Copy

• Assigning objects
  – An object can be assigned to another object of the same type using the assignment operator (=)
  – Member by member copy

• Objects may be
  – Passed as function arguments
  – Returned from functions (call-by-value default)
// Fig. 6.12: fig06_12.cpp
// Demonstrating that class objects can be assigned
// to each other using default memberwise copy
#include <iostream>

using std::cout;
using std::endl;

// Simple Date class
class Date {
    public:
    Date( int = 1, int = 1, int = 1990 ); // default constructor
    void print();

    private:
    int month;
    int day;
    int year;
};

// Simple Date constructor with no range checking
Date::Date( int m, int d, int y )
{    month = m;
    day = d;
    year = y;
}

// Print the Date in the form mm-dd-yyyy
void Date::print()
{    cout << month << '-' << day << '-' << year; }

© 2000 Prentice Hall, Inc. All rights reserved.
```cpp
int main()
{
    Date date1(7, 4, 1993), date2;   // d2 defaults to 1/1/90
    cout << "date1 = ";
    date1.print();
    cout << "\ndate2 = ";
    date2.print();

    date2 = date1;   // assignment by default memberwise copy
    cout << "\n\nAfter default memberwise copy, date2 = ";
    date2.print();
    cout << endl;

    return 0;
}
```

Program Output:

```
date1 = 7-4-1993
date2 = 1-1-1990

After default memberwise copy, date2 = 7-4-1993
```
6.17 Software Reusability

- Software reusability
  - Implementation of useful classes
  - Class libraries exist to promote reusability
    - Allows for construction of programs from existing, well-defined, carefully tested, well-documented, portable, widely available components
  - Speeds development of powerful, high-quality software