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

16 // Fig. 6.9: create.cpp
17 // Member function definitions for class CreateAndDestroy
18 #include <iostream>
19
20 using std::cout;
21 using std::endl;
22
23 #include "create.h"
24
25 CreateAndDestroy::CreateAndDestroy( int value )
26 {
27    data = value;
28    cout << "Object " << data << " constructor" << endl;
29 }
30
31 CreateAndDestroy::~CreateAndDestroy()
32 { cout << "Object " << data << " destructor" << endl; }
3. Create multiple objects of varying types

```cpp
// Fig. 6.9: fig06_09.cpp
// Demonstrating the order in which constructors and destructors are called.
#include <iostream>

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

#include "create.h"

void create( void ); // prototype

CreateAndDestroy first( 1 ); // global object

void create( void )
{
    CreateAndDestroy fifth( 5 );
    cout << " (local automatic in create)" << endl;
    static CreateAndDestroy sixth( 6 );
    cout << " (local static in create)" << endl;
    cout << " (local automatic in create)" << endl;
    CreateAndDestroy seventh( 7 );
    cout << " (local automatic in create)" << endl;
}

int main()
{
    cout << " (global created before main)" << endl;

    CreateAndDestroy second( 2 ); // local object
    cout << " (local automatic in main)" << endl;
    static CreateAndDestroy third( 3 ); // local object
    cout << " (local static in main)" << endl;
    create(); // call function to create objects
    CreateAndDestroy fourth( 4 ); // local object
    cout << " (local automatic in main)" << endl;
    return 0;
}
```

**OUTPUT**

Object 1   constructor   (global created before main)
Object 2   constructor   (local automatic in main)
Object 3   constructor   (local static in main)
Object 5   constructor   (local automatic in create)
Object 6   constructor   (local static in create)
Object 7   constructor   (local automatic in create)
Object 7   destructor
Object 5   destructor
Object 4   destructor
Object 2   destructor
Object 6   destructor
Object 3   destructor
Object 1   destructor

Notice how the order of the constructor and destructor call depends on the types of variables (automatic, global, and static) they are associated with.
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 lvalue.
  - Reference can receive a value, which changes the original object as well

- Returning references
  - public member functions can return non-const references to private data members
  - Should be avoided, breaks encapsulation

- Please avoid using references in this way, very, very bad!!!
// Fig. 6.11: time4.h
// Declaration of the Time class.
// Member functions defined in time4.cpp

#ifndef TIME4_H
#define TIME4_H

class Time {
public:
    Time( int = 0, int = 0, int = 0 );
    void setTime( int, int, int );
    int getHour();
    int &badSetHour( int ); // DANGEROUS reference return
private:
    int hour;
    int minute;
    int second;
};
#endif

#include "time4.h"

// Fig. 6.11: time4.cpp
// Member function definitions for Time class.

// 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
}
53 // Fig. 6.11: fig06_11.cpp
54 // Demonstrating a public member function that
55 // returns a reference to a private data member.
56 // Time class has been trimmed for this example.
57 #include <iostream>
58
59 using std::cout;
60 using std::endl;
61 #include "time4.h"
62
63 int main()
64 {
65    Time t;
66    int &hourRef = t.badSetHour( 20 );
67
68    cout << "Hour before modification: " << hourRef;
69    hourRef = 30;  // modification with invalid value
70    cout << "Hour after modification: " << t.getHour();
71
72    // Dangerous: Function call that returns
73    // a reference can be used as an lvalue!
74    t.badSetHour(12) = 74;
75    cout << "\n" << "POOR PROGRAMMING PRACTICE!!!!!!!!\n"
76    << "badSetHour as an lvalue, Hour: "
77    << t.getHour() << endl;
78
79    return 0;
80 }

© 2000 Prentice Hall, Inc. All rights reserved.
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)

```cpp
// 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;

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; }
```
```cpp
int main()
{
    Date date1(7, 4, 1993), date2; // d2 defaults to 1/1/90
    cout << "date1 = ";
    date1.print();
    cout << "date2 = ";
    date2.print();

    date2 = date1; // assignment by default memberwise copy
    cout << "After 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