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 the 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

Outline

1. Create a header file
  1.1 Include function prototypes for the destructor and constructor

2. Load the header file
  2.1 Modify the constructor and destructor

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 CreateAndDestroy::CreateAndDestroy( int value )
24 {
25    data = value;
26    cout << "Object " << data << "   constructor" << endl;
27 }
28
29 CreateAndDestroy::~CreateAndDestroy()
30 {
31    cout << "Object " << data << "   destructor " << endl;
32    delete this;
33    cout << "Destructor and Destructors changed to print when they are called."
34    delete this;
35  }
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 it 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 value
  - 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!!!
### 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
#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;
    }

Date::Date( int m, int d, int y )
{    month = m;
    day = d;
    year = y;
}

void Date::print()
{    cout << month << '-' << day << '-' << year;
}

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() << " 

    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