More on Classes

CS 1: Problem Solving & Program Design Using C++

Objectives

- Assign ourselves the task of assignment
- Point to pointers as class members
- Find out some additional class features
- Finally, look at common programming errors

Assignment

- MEMBERWISE ASSIGNMENT: allows assignment of data member values of an object to their counterparts in another object of the class
- Compiler builds this type of default assignment if there are no instructions to the contrary
- Assignment operators:
    - Declared in class declaration section
    - Defined in class implementation section
    - EXAMPLE: a = b;

Assignment Example: Declaration

```
#include <iostream>
#include <iomanip>
using namespace std;

// class declaration
class Date
{
  private:
    int month;
    int day;
    int year;
  public:
    Date(int = 7, int = 4, int = 2006); // constructor prototype
    void showDate(); // member function to display a Date
};
```

Assignment Example: Implementation

```
// implementation section
Date::Date(int mm, int dd, int yyyy)
{
  month = mm;
  day = dd;
  year = yyyy;
}

void Date::showDate()
{
  cout << setfill ('0')
          << setw(2) << month << '/'
          << setw(2) << day << '/'
          << setw(2) << year % 100;
  return;
}
```

Assignment Example: main()

```
int main()
{
  Date a(4,1,1999), b(12,18,2006);  // declare two objects

  cout << "\nThe date stored in a is originally ";
  a.showDate();     // display the original date
  a = b;            // assign b's value to a

  cout << "\nAfter assignment the date stored in a is ";
  a.showDate();     // display a's values
  cout << endl;

  return 0;
}
```

Assignment Operator Declaration

- FORMAT: void operator=(Date&);
  - Declares simple assignment operator for Date class
  - Add to public section of class declarations
- KEYWORD: void, as assignment returns no value
- operator= indicates overloading of assignment operator with new version
- (className&): argument to operator is class reference

Assignment Operator Implementation Format

- Assignment operator implementation format:

```
void Date::operator=(Date& newdate)
{
  day = newdate.day;     // assign the day
  month = newdate.month; // assign the month
  year = newdate.year;  // assign the year
}
```

- newdate: a reference to the Date class
    - Reference parameters facilitate overloaded operators
- day, month, and year members of newdate: assigned to corresponding members of current object

Other Issues Affecting Assignment Operators

- Use constant reference parameter
  - FORMAT: void Date::operator=(const Date& secdate);
  - Precludes inadvertent change to secdate
- Assignment returns no value
  - Cannot be used in multiple assignments such as:

    a = b = c
  - Reason: a = b = c equivalent to a = (b + c)
    - But (b + c) returns no value making assignment to a an error

Copy Constructors

- COPY CONSTRUCTOR: Initializes an object using another object of same class
- EXAMPLE: Two equivalent formats

```
Date b = a;

Date b(a);
```

- DEFAULT COPY CONSTRUCTOR: constructed by compiler if none declared by programmer
  - Similar to default assignment constructor
  - Performs memberwise copy between objects
  - Does not work well with pointer data members

Copy Constructors (2)

- FORMAT: className(const className&);
    - Function name must be class name
    - Parameter is reference to class
    - Parameter specified as const to prevent inadvertent change
- DECLARATION: Copy constructor for Date class

```
Date(const Date&);
```

Copy Constructor Implementation

```
Date::Date(const Date& olddate)
{
  month = olddate.month;
  day = olddate.day;
  year = olddate.year;
}
```

Base/Member Initialization

- Copy constructor does not perform true initialization
  - Creates and then assigns
- Base/Member initialization list: Initializes an object with no assignment
  - List can be applied only to constructor functions

Base/Member Initialization List Construction Method #1

- Construct list in class declaration section:

```
    public:
      Date(int mo = 7, int da = 4, int yr = 2006): month(mo),
day(da), year (yr) { }
```

Base/Member Initialization List Construction Method #2

- Declare prototype in declaration section and create list in implementation section

```
// class declaration section
public:
  Date(int = 7, int = 4, int = 2006);  // prototype
// class implementation section
Date::Date(int mo, int da, int yr) : month(mo),
day(da),year(yr) {}
```

Pointers as Class Members Declaration

```
#include <iostream>
#include <iomanip>
using namespace std;

// class declaration
class Test
{
  private:
    int idNum;
    double *ptPay;
  public:
    Test(int = 0, double * = NULL); // constructor
    void setvals(int, double *);  // access function
    void display();   // access function
};
```

Pointers as Class Members Declaration

```
Test::Test(int id, double *pt)
{
  idNum = id;
  ptPay = pt;
}

void Test::setvals(int a, double *b)
{
  idNum = a;
  ptPay = b;
}

void Test::display()
{
  cout << "\nEmployee number " << idNum << " was paid $"
          << setiosflags(ios::fixed) << setiosflags(ios::showpoint)
          << setw(6) << setprecision(2) << *ptPay << endl;
 }
```

Pointers as Class Members main()

```
int main()
{
  Test emp;
  double pay = 456.20;
  emp.setvals(12345, &pay);
  emp.display();
  return 0;
}
```

- OUTPUT: Employee number 12345 was paid $456.20

Pointers as Class Members: Actions Performed

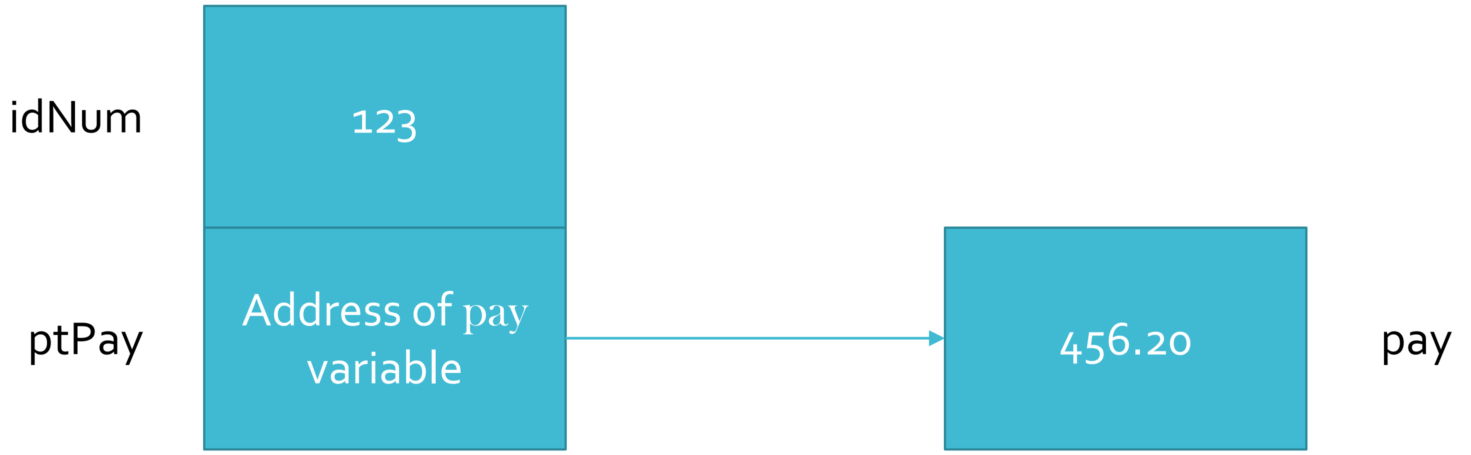
- Test() constructor initializes:
  - idNum data member to first parameter: 0
  - ptPay pointer member to second parameter: NULL
- display() function: outputs value pointed to by pointer member, ptPay
  - Pointer member used like any other pointer variable
- setvals() function: alters member values after object is declared
  - First parameter (an integer) is assigned to idNum
  - Second parameter (an address) is assigned to ptPay

Pointers as Class Members: Actions Performed (2)

- main() function: creates emp object
  - emp initialized using constructor's default arguments
- setvals() function: assigns value 12345 and address of variable pay to emp object data members
- display() function: displays value whose address stored in emp.ptPay

Storing An Address in a Data Member

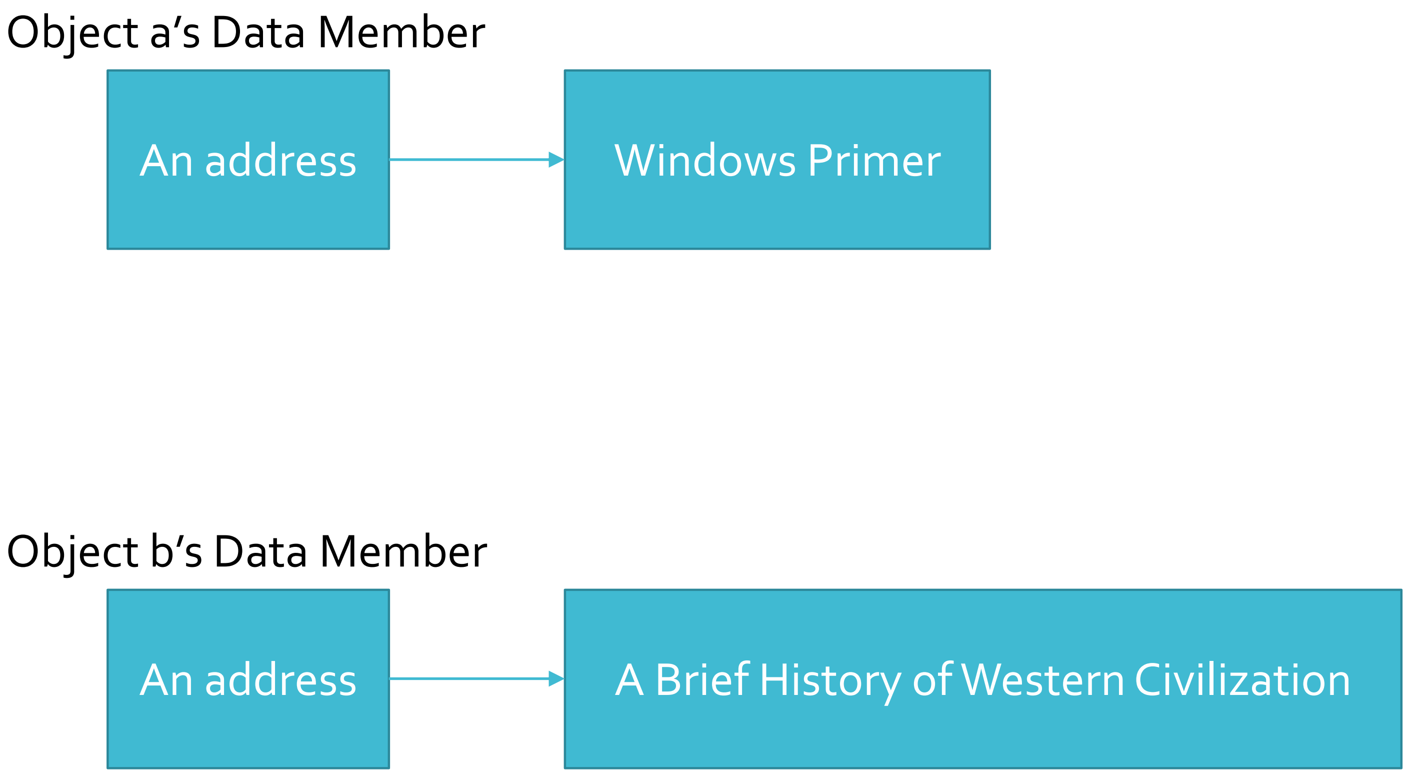
Object emp's data members:



Pointers as Class Members: Example of Pointer Use

- Store list of book titles
  - Inefficient to use fixed-length array
  - Use pointer member to character array
    - Allocate array of correct size for each book title as needed
  - This arrangement illustrated in the following figure

Two Objects That Contain Pointer Data Members



List of Book Titles Example: Declaration

```
#include <iostream>
#include <string>
using namespace std;

class Book
{
  private:
    char *title; // a pointer to a book title
  public:
    Book(char * = '\0'); // constructor
    void showtitle(void); // display the title
};
```

List of Book Titles Example: Implementation

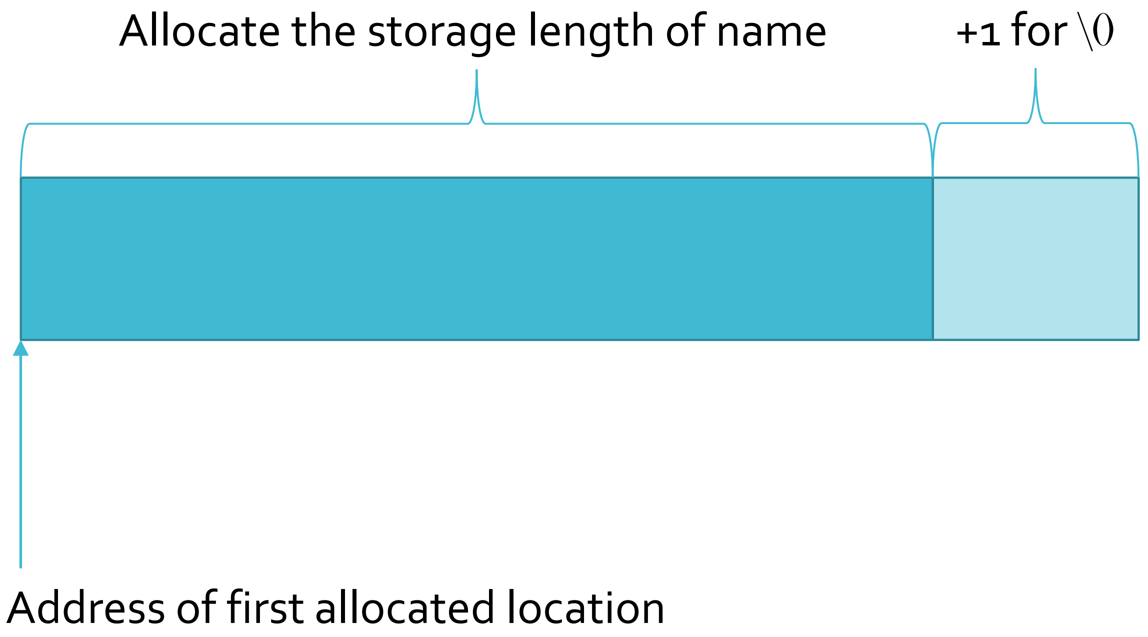
```
// class implementation
Book::Book(char *strng)
{
  title = new char[strlen(strng)+1];      // allocate memory
  strcpy(title,strng);                    // store the string
}

void Book::showtitle(void)
{
  cout << title << endl;
}
```

List of Book Titles Example: main()

```
int main()
{
  Book book1("Windows Primer"); // create 1st title
  // 2nd title
  Book book2("A Brief History of Western Civilization");
  book1.showtitle(); // display book1's title
  book2.showtitle(); // display book2's title
  return 0;
}
```

Allocating Memory for title

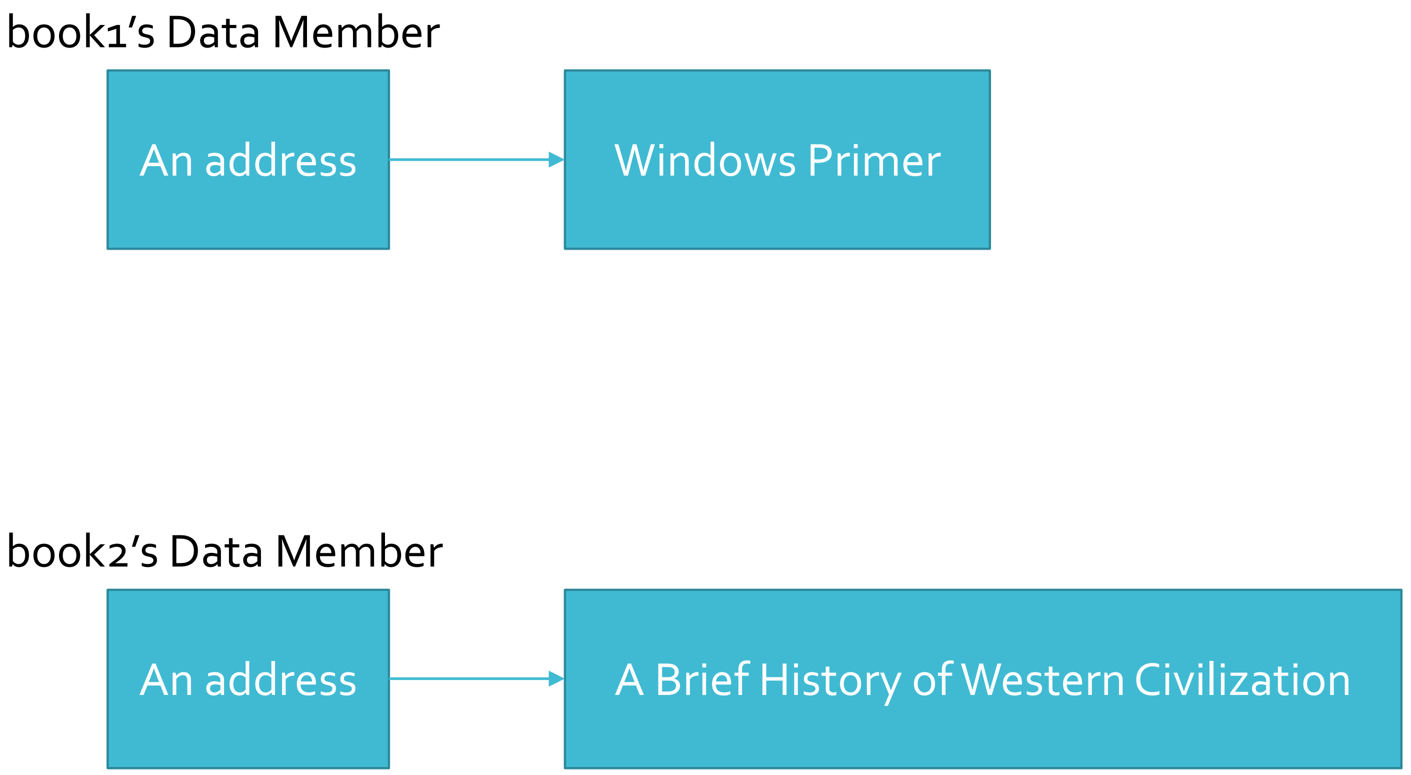
```
title = new char[strlen(strng)+1];
```


Allocate the storage length of name
+1 for \0
Address of first allocated location

Assignment Operators and Copy Constructors Reconsidered

- If a class contains no pointer data members, compiler-provided defaults work well for assignment operators and copy constructors
- Compiler defaults provide member-by-member operation with no adverse side effects
- Problems occur with defaults if pointers involved

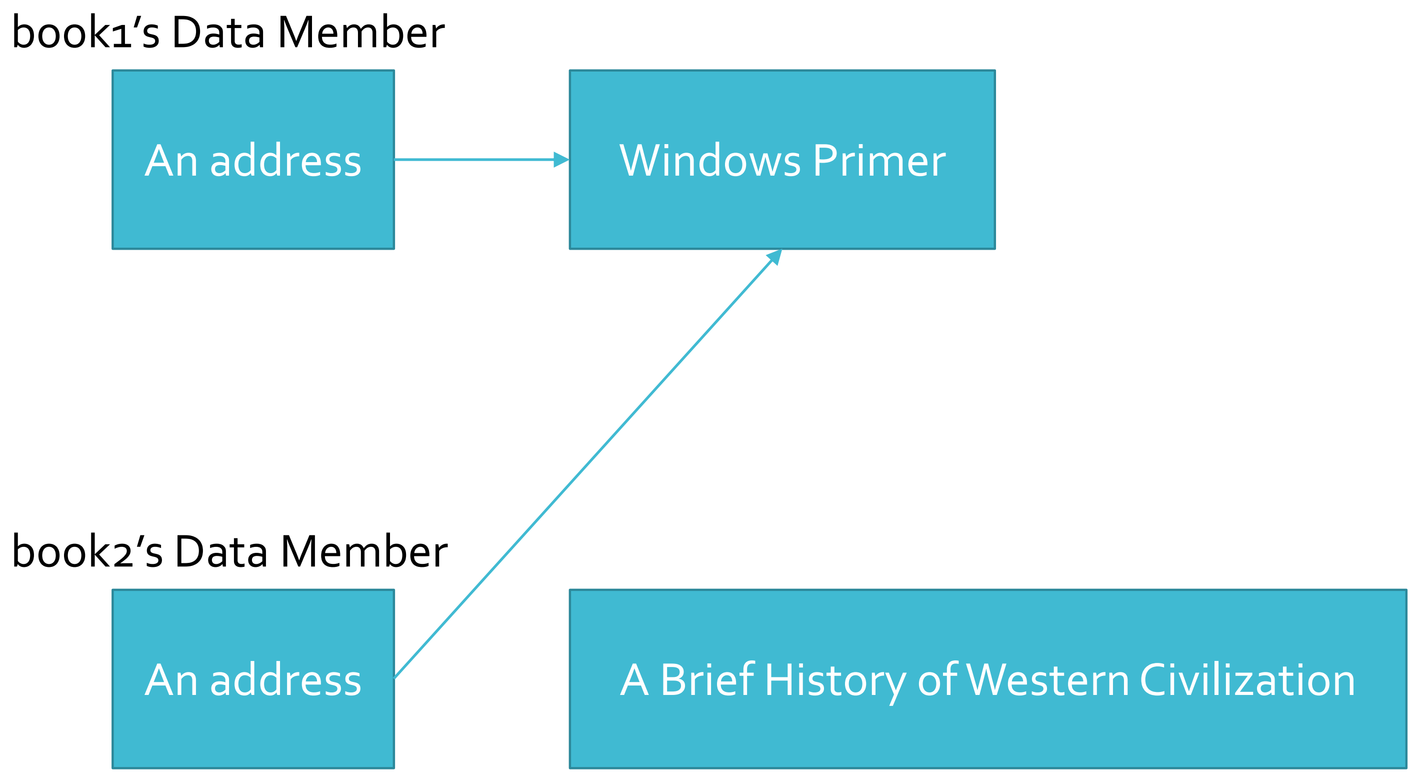
Before the Assignment book2 = book1



Now, Let's Assign One Book to the Other

- Now insert statement: book 2 = book1; before closing brace of main()
- Compiler default assignment is used
- Produces memberwise copy
  - Address in book1's pointer copied into book2's pointer
  - Both pointers now point to same address
  - Address of A Brief History of Western Civilization is lost

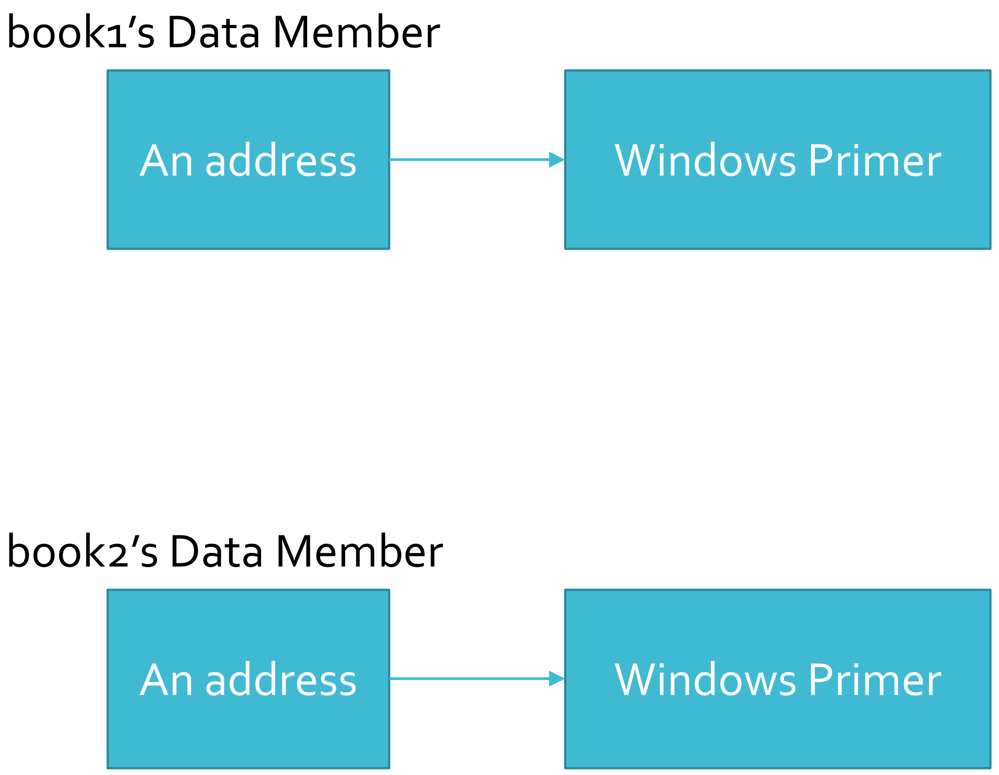
After the Assignment book2 = book1



Oops, There's a Problem; How Do We Solve It?

- Effect of loss of address of: A Brief History of Western Civilization
  - No way for operating system to release this memory (until program terminates)
  - If destructor attempts to release memory pointed to by book2, book1 will point to an undefined memory location
- Solution: copy book titles and leave pointers alone
  - Must write our own assignment operator

Here's What We Really Want…



Assignment Operator for Pointers

- Format

```
void Book::operator=(Book& oldbook)
{
  if(oldbook.title != NULL) // check that it exists
    delete(title);      // release existing memory
  title = new char[strlen(oldbook.title) + 1]; // allocate new memory
  strcpy(title, oldbook.title); // copy the title
}
```

- Problems associated with assignment operator also exist with default copy constructor
  - Need to also define a new copy constructor

Class Scope

- Names of data and function members are local to scope of class
- If global variable name reused within class, global variable is hidden by class data member
- Member function names are local to class they are declared in
- Local function variables hide names of class data members that have same name

static Members

- As each class created, it gets its own block of memory for data members
- For every instantiation, we may want to share same memory location for specific variable
- EXAMPLE: in class of employee records each employee subject to same state sales tax
  - Could make sales tax a global variable but this violates principles of data hiding
  - Better option: Declare tax as a static class variable
    - Static data members act as global class variables

The this Pointer

- Each time an object is created, distinct area of memory is set aside for its data members
- No replication of memory for member functions
  - For each class, only one copy of member function is retained in memory
  - Each object uses same functions

The this Pointer (2)

- Sharing member functions: requires identification of data structure to be operated on
  - Accomplished by providing address information to function indicating location of object's data structure
  - EXAMPLE: statement a.showDate() passes address of object a into showDate() member function
- Question: Where is address of object a stored and how is it passed to showDate()
- this: special pointer variable
  - Automatically supplied as hidden argument to each nonstatic member function that is called

friend Functions

- Accessing and manipulating class’s private data members: done only through member functions
- Sometime necessary to provide such access to nonmember functions
- FRIEND FUNCTIONS: list of nonmember functions that are granted same privileges as members
- FRIENDS LIST: series of function prototypes preceded by word friend and included in class’s declaration section
  - EXAMPLE: friend double addreal(complex&, complex&);

Common Programming Errors

- Using default copy constructors and default assignment operators with classes that contain pointer values
- Using user-defined assignment operator in multiple assignment expression when operator has not been defined to return an object
- Using keyword static when defining either a static data or function member

Common Programming Errors (2)

- Using keyword friend when defining a friend function
    - friend keyword should be used only within class declaration section
- Failing to instantiate static data members before creating class objects that must access these data members
- Forgetting that this is pointer that must be dereferenced using either *this or this->

Summary

- Assignment operator: may be declared with the function prototype:
  - void operator=(className&);
- Copy constructor: one object is initialized using another object of the same class:
  - className(const className&);
- Pointers: may be included as class members
  - Adhere to same rules as pointer variables

Summary (2)

- Copy default constructors and default assignment operators typically not useful with classes that contain pointer members
- Class scope: data and function members are local to scope of their class
- For each class object, separate set of memory locations is reserved for all data members except those declared as static

Summary (3)

- Static function members apply to class as whole, rather than individual objects
  - Static function members can only access static data members and other static function members
- this: pointer argument used to pass address of an object's data member to member functions
- friend: class declaration that allows nonmember function to access class private data members