

LAB 5: Data Member Initialization & Class Composition



Employee-Date Class Composition

Rung-Bin Lin

International Bachelor Program in Informatics
Yuan Ze University

3/25/2021

Purposes of the Lab

- **const** objects and **const** member functions (Chapter 10.1)
- Data member initialization (Chapter 10.2)
- Class composition (Chapter 10.3)
 - A class uses the definitions from another classes as **types** for defining its members.
- Class Composition has a “has-a” relationship between the underlying two classes. That is, one class has the other class as a member.
- *this* pointer (Chapter 10.5)

const Objects and const Functions

- **const object** must be initialized during an object being created. After that it can not be modified.
- **const object** can be only processed by constant member functions

```
const Time noon(12, 0, 0); // const object
void printUniversal() const; // prototype of a const member function
void Time::printUniversal() const
{
    cout << setfill( '0' ) << setw( 2 ) << hour << ":"
    << setw( 2 ) << minute << ":" << setw( 2 ) << second;
} // end function printUniversal
```

How to Initialize const Object

• Use data member initializer

```
class Increment
{
public:
    Increment( int c = 0, int i = 1 ); // default constructor

    // function addIncrement definition
    void addIncrement()
    {
        count += increment;
    } // end function addIncrement

    void print() const; // prints count and increment
private:
    int count;
    const int increment; // const data member
}; // end class Increment
```

```
#endif
```

```
// constructor
Increment::Increment( int c, int i )
    : count( c ), // initializer for non-const member
    increment( i ) // required initializer for const member
{
    // empty body
} // end constructor Increment

// print count and increment values
void Increment::print() const
{
    cout << "count = " << count << ", increment = " << increment
    << endl;
} // end function print
```

Order of Data Member Initialization

- **Data members are usually initialized in the order in which they appear in the class definition.** The first data member is initialized first, the second is next, etc. The order often does not matter. **However, if initialization of one data member depends on the other, the order of data members in class definition matters.**

For example:

```
class Stack {  
    private:  
        int size;  
        int top;  
        int *stackPtr;
```

```
    public:
```

```
        Stack(int s):size( s > 0 ? s : 10 ), top( -1 ),  
        stackPtr( new int[ size ] ) { ...}
```

```
// If declared as follows, compilation will fail  
// because initialization of stackPtr will be done  
// first, but it is initialized with undefined size.  
int *stackPtr;  
int size;  
int top;
```

Destructor ~

- **Destructor is a member function destructing or deleting an object. (see an example in Fig. 10.3)**
 - Destructors have same name as the class preceded by a tilde (~)
 - Destructors don't take any argument and don't return anything
- **A destructor function is called automatically when the object goes out of scope.**
 - the function ends
 - the program ends
 - a block containing local variables ends
 - a delete operator is called

Class Date

```
class Date
{
public:
    static const int monthsPerYear = 12; // number of months in a year
    Date( int = 1, int = 1, int = 1900 ); // default constructor
    void print() const; // print date in month/day/year format
    ~Date(); // provided to confirm destruction order
private:
    int month; // 1-12 (January-December)
    int day; // 1-31 based on month
    int year; // any year

    // utility function to check if day is proper for month and year
    int checkDay( int ) const;
}; // end class Date
```

Class Composition: Employee uses Date

```
#include <string>
#include "Date.h" // include Date class definition
using namespace std;

class Employee
{
public:
    Employee( const string &, const string &, const Date &, const Date & );
    void print() const;
    ~Employee(); // provided to confirm destruction order
private:
    string firstName; // composition: member object
    string lastName; // composition: member object
    const Date birthDate; // composition: member object
    const Date hireDate; // composition: member object
}; // end class Employee
```


“this” Pointer

- Every object has access to its own address through a pointer called **this**.
 - The **this** pointer is not part of the object itself.
 - It is created and passed by the compiler as an **implicit** argument to each of the **object's non-static member functions**.
 - It can be used in a member function implicitly or explicitly.

```
class Test
{
public:
    // default constructor
    Test( int = 0 );
    void print() const;
private:
    int x;
}; // end class Test
```

```
void Test::print() const {
    // implicitly use the this pointer to access the member x
    cout << "    x = " << x;
    // explicitly use the this pointer and the arrow operator
    // to access the member x
    cout << "\n this->x = " << this->x;
    // explicitly use the dereferenced this pointer and
    // the dot operator to access the member x
    cout << "\n(*this).x = " << ( *this ).x << endl;
} // end function print
```

Using the **this** Pointer for Cascading Function Calls

- A member function should either return **a reference** to a class object or **a pointer** to a class object for enabling such a capability. (see Chapter 10.5 for details)

```
class Time {  
public:  
    Time( int = 0, int = 0, int = 0 ); // default constructor  
    // set functions (the Time & return types enable cascading)  
    Time &setTime( int, int, int ); // set hour, minute, second  
    Time &setHour( int ); // set hour  
    Time &setMinute( int ); // set minute  
    Time &setSecond( int ); // set second  
private:  
    int hour; // 0 - 23 (24-hour clock format)  
    int minute; // 0 - 59  
    int second; // 0 - 59  
}; // end class Time
```

```
Time &Time::setHour(int h)  
{  
    hour = (h>=0 && h<24)? h : 0;  
    return *this;  
}
```

```
int main()  
{  
    Time t; // create Time object  
  
    // cascaded function calls  
  
    t.setHour( 18 ).setMinute( 30 ).setSecond( 22 );  
}
```

Type of the **this** Pointer

- The type of the **this** pointer depends on the type of the object and whether the member function in which **this** is used. For example,
 - In a nonconstant member function of class Time such as
`void printUniversal();`
the **this** pointer has type **Time * const** (a constant pointer to a nonconstant Time object).
 - In a constant member function of class Time such as
`void printUniversal() const;`
the **this** pointer has type **const Time * const** (a constant pointer to a constant Time object).

Type of the **this** Pointer Matters

```
Time &Time::setHour(int h)
```

```
{  
    hour = (h>=0 && h<24)? h : 0;  
    return *this;  
}
```

```
const Time &Time::printUniversal_c() const
```

```
{  
    cout << setfill( '0' ) << setw( 2 ) << hour << ":"  
        << setw( 2 ) << minute << ":" << setw( 2 ) <<  
    second;  
    return *this; // return a constant Time object  
} // end function printUniversal
```

```
int main()
```

```
{  
    const Time noon(12, 0, 0);  
    Time T1(12, 12, 12);  
    T1.printUniversal().setHour(20); // OK  
    T1.printUniversal_c().setHour(20); // not OK  
    noon.printUniversal_c().printUniversal_c(); //OK  
    noon.printUniversal().setHour(20); // not OK  
}
```

```
Time &Time::printUniversal()
```

```
{  
    cout << setfill( '0' ) << setw( 2 ) << hour << ":"  
        << setw( 2 ) << minute << ":" << setw( 2 ) <<  
    second;  
    return *this; // return a noncontant Time object  
} // end function printUniversal
```

Lab 5 : Employee & Date Class Composition

- Do the following tasks by modifying the code in Fig.10.10 ~Fig. 10.13 .
 - Modify the Employee class to include a data member called **dismissDate** which is the date of leaving the company. When an Employee is hired, dismissDate is set to year=1900, month=1, day=1.
 - Also include an *int* data member monSalary which is the salary per month.
 - Add a member function eDismiss(const Date &) to set the dismiss date of an employee. The dismiss date should be later than hire date. If not, “First Last with inconsistent dates!” should be printed, where First is the first name and Last is the last name of the employee.
 - Add a member function increaseSalary(int xxx) to Employee class to increase the salary of an employee by xxx dollars.
 - Modify the constructor of Employee to check the consistency of birthDate, hireDate, and dismissDate. That is, hireDate should be later than birthDate, and dismissDate should be later than hireDate except when dismissDate is year=1900, month=1, day=1. If any inconsistency is found, print a message “First Last with inconsistent dates!”, where First is the first name and Last is the last name of the employee.
 - Modify or add any member functions of Date and Employee classes that will make the main() function work correctly and generate the output exactly same as that given in the example output.

main() Function of Employee (should not be modified)

```
int main()
{
    Date birth( 3, 24, 1961 );
    Date hire( 4, 12, 1998 );
    Date dismiss(2, 28, 2021);
    Date zero;
    Employee manager1( "Tom", "Crouse", birth, hire, zero, 66000 );
    cout << endl;
    manager1.print();
    manager1.increaseSalary(2000).print();
    manager1.eDismiss(dismiss)->print();
    cout << "\nTest Date constructor with invalid values:\n";
    Date lastDayOff( 14, 35, 1994 ); // invalid month and day
    cout << endl;
    Date birth1( 1, 1, 1968 );
    Date hire1( 1, 1, 1994 );
    Date dismiss1(3, 28, 1990);
    Employee E1( "Mary", "Hunton", birth1, hire1, dismiss1, 45000 );
    cout << endl;
    Date dismiss2(3,28, 2000);
    E1.eDismiss(dismiss2)->print();
    E1.increaseSalary(-2000).print();
} // end main
```

Output

```
Date object constructor for date 3/24/1961
Date object constructor for date 4/12/1998
Date object constructor for date 2/28/2021
Date object constructor for date 1/1/1900
Employee object constructor: Tom Crouse Monthly paid: 66000

TomCrouse Hired: 4/12/1998 Birthday: 3/24/1961 Monthly pay: 66000 Dismissing: 1/1/1900
TomCrouse Hired: 4/12/1998 Birthday: 3/24/1961 Monthly pay: 68000 Dismissing: 1/1/1900
TomCrouse Hired: 4/12/1998 Birthday: 3/24/1961 Monthly pay: 68000 Dismissing: 2/28/2021

Test Date constructor with invalid values:
Invalid month (14) set to 1.
Invalid day (35) set to 1.
Date object constructor for date 1/1/1994

Date object constructor for date 1/1/1968
Date object constructor for date 1/1/1994
Date object constructor for date 3/28/1990
Employee object constructor: Mary Hunton Monthly paid: 45000
Mary Hunton with inconsistent dates!

Date object constructor for date 3/28/2000
MaryHunton Hired: 1/1/1994 Birthday: 1/1/1968 Monthly pay: 45000 Dismissing: 3/28/2000
MaryHunton Hired: 1/1/1994 Birthday: 1/1/1968 Monthly pay: 43000 Dismissing: 3/28/2000
Date object destructor for date 3/28/2000
Employee object destructor: Hunton, Mary
Date object destructor for date 3/28/2000
Date object destructor for date 1/1/1994
Date object destructor for date 1/1/1968
Date object destructor for date 3/28/1990
Date object destructor for date 1/1/1994
Date object destructor for date 1/1/1968
Date object destructor for date 1/1/1994
Employee object destructor: Crouse, Tom
Date object destructor for date 2/28/2021
Date object destructor for date 4/12/1998
Date object destructor for date 3/24/1961
Date object destructor for date 1/1/1900
Date object destructor for date 2/28/2021
Date object destructor for date 4/12/1998
Date object destructor for date 3/24/1961
```

with 7 destructors

with 7 destructors

```

// Fig. 10.10: Date.h
// Date class definition; Member functions defined in Date.cpp
#ifndef DATE_H
#define DATE_H

class Date
{
public:
    static const int monthsPerYear = 12; // number of months in a year
    Date( int = 1, int = 1, int = 1900 ); // default constructor
    void print() const; // print date in month/day/year format
    ~Date(); // provided to confirm destruction order
private:
    int month; // 1-12 (January-December)
    int day; // 1-31 based on month
    int year; // any year

    // utility function to check if day is proper for month and year
    int checkDay( int ) const;
}; // end class Date
#endif

```



```

// Fig. 10.11: Date.cpp
// Date class member-function definitions.
#include <iostream>
#include "Date.h" // include Date class definition
using namespace std;

// constructor confirms proper value for month; calls
// utility function checkDay to confirm proper value for day
Date::Date( int mn, int dy, int yr )
{
    if ( mn > 0 && mn <= monthsPerYear ) // validate the month
        month = mn;
    else
    {
        month = 1; // invalid month set to 1
        cout << "Invalid month (" << mn << ") set to 1.\n";
    } // end else

    year = yr; // could validate yr
    day = checkDay( dy ); // validate the day

    // output Date object to show when its constructor is called
    cout << "Date object constructor for date ";
    print();
    cout << endl;
} // end Date constructor

```

```
// print Date object in form month/day/year
void Date::print() const
{
    cout << month << '/' << day << '/' << year;
} // end function print

// output Date object to show when its destructor is called
Date::~~Date()
{
    cout << "Date object destructor for date ";
    print();
    cout << endl;
} // end ~Date destructor
```

```

// utility function to confirm proper day value based on
// month and year; handles leap years, too
int Date::checkDay( int testDay ) const
{
    static const int daysPerMonth[ monthsPerYear + 1 ] =
        { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

    // determine whether testDay is valid for specified month
    if ( testDay > 0 && testDay <= daysPerMonth[ month ] )
        return testDay;

    // February 29 check for leap year
    if ( month == 2 && testDay == 29 && ( year % 400 == 0 ||
        ( year % 4 == 0 && year % 100 != 0 ) ) )
        return testDay;

    cout << "Invalid day (" << testDay << ") set to 1.\n";
    return 1; // leave object in consistent state if bad value
} // end function checkDay

```

```

// Fig. 10.12: Employee.h
// Employee class definition showing composition.
// Member functions defined in Employee.cpp.
#ifndef EMPLOYEE_H
#define EMPLOYEE_H

#include <string>
#include "Date.h" // include Date class definition
using namespace std;

class Employee
{
public:
    Employee( const string &, const string &,
             const Date &, const Date & );
    void print() const;
    ~Employee(); // provided to confirm destruction order
private:
    string firstName; // composition: member object
    string lastName; // composition: member object
    const Date birthDate; // composition: member object
    const Date hireDate; // composition: member object
}; // end class Employee

#endif

```

```

// Fig. 10.13: Employee.cpp
// Employee class member-function definitions.
#include <iostream>
#include "Employee.h" // Employee class definition
#include "Date.h" // Date class definition
using namespace std;

// constructor uses member initializer list to pass initializer
// values to constructors of member objects
Employee::Employee( const string &first, const string &last,
    const Date &dateOfBirth, const Date &dateOfHire )
    : firstName( first ), // initialize firstName
      lastName( last ), // initialize lastName
      birthDate( dateOfBirth ), // initialize birthDate
      hireDate( dateOfHire ) // initialize hireDate
{
    // output Employee object to show when constructor is called
    cout << "Employee object constructor: "
        << firstName << ' ' << lastName << endl;
} // end Employee constructor

```

```
// print Employee object
void Employee::print() const
{
    cout << lastName << ", " << firstName << " Hired: ";
    hireDate.print();
    cout << " Birthday: ";
    birthDate.print();
    cout << endl;
} // end function print

// output Employee object to show when its destructor is called
Employee::~~Employee()
{
    cout << "Employee object destructor: "
        << lastName << ", " << firstName << endl;
} // end ~Employee destructor
```