

# 객체지향프로그래밍 II



## Lecture 2

### Class의 이해 및 활용 (교과서 9장, Part 3)

1. Destructors
2. 생성자와 소멸자의 호출시기
3. Default Memberwise Assignment
4. Copy Constructor





# 1. Destructors



## 클래스 소멸자 (class destructors)

- ✓ 특별한 용도의 멤버 함수
- ✓ 소멸자의 이름: ‘~’에 클래스의 이름을 붙임
  - 예) ~Time
- ✓ 객체가 소멸 될 때 무조건 호출된다.
  - 예) 자동변수로 선언된 객체가 소멸될 때
- ✓ 객체 생성시 할당된 메모리를 해제하지 않는다.
  - 종료 정리 작업 (termination housekeeping)을 수행
  - 소멸자 실행 후, 시스템이 객체의 메모리를 회수
    - ➡ 따라서 메모리는 재사용 될 수 있게 된다.



## 소멸자 (destructors)

✓ 매개변수를 받지도 않고 값을 반환하지도 않는다.

- 반환형이 없다. — `void` 형도 아님

✓ 클래스는 단 한 개의 소멸자를 가질 수 있다.


- 소멸자 오버로딩(overloading)은 허락되지 않는다.

✓ 만약 프로그래머가 명시적으로 소멸자를 지정하지 않았다면, 컴파일러는 “빈(empty)” 소멸자를 만든다.

- 아무 일도 하지 않음



## 2. 생성자와 소멸자의 호출시기





## 생성자와 소멸자의 호출 메커니즘

- ✓ 객체 생성과 소멸시 내부적으로 (implicitly) 자동 호출됨
- ✓ 생성자와 소멸자의 호출 시기와 순서는 어느 scope에서 객체가 생성되느냐에 따라 다름
  - 예제 참조
- ✓ 일반적으로 생성자의 호출 순서의 역순으로 소멸자가 호출됨



# 생성자와 소멸자의 호출 시기

☞ 객체가 **global scope**에서 정의될 때 (즉, 전역 변수일 때)

- ✓ 전역 변수는 프로그램 수행 내내 살아있음
- ✓ **main** 함수가 시작되기 전에 객체가 생성되어 생성자 호출
- ✓ **main** 함수가 종료되면 객체가 소멸되어 소멸자 호출

☞ 객체가 자동 지역 변수 (**local automatic variable**) 일 때

- ✓ 객체가 정의될 때 객체가 생성되어 생성자 호출
- ✓ 객체가 포함된 **scope**를 벗어날 때 소멸되어 소멸자 호출



# 생성자와 소멸자의 호출 시기

- ☞ 객체가 정적 지역 변수 (static local variable) 일 때
  - ✓ Static 변수는 단 한번 생성되어 프로그램 수행 내내 살아있는 변수이다
  - ✓ 따라서 생성자는 객체가 처음 생성될 때 단 한번 호출
  - ✓ main 함수가 종료되면 객체가 소멸되어 소멸자 호출
- ☞ 전역 객체와 정적 객체는 그 생성 순서의 역순으로 소멸됨



# 생성자와 소멸자의 호출 시기 예제 (CreatAndDestroy.h)

```
1 // Fig. 9.11: CreateAndDestroy.h
2 // Definition of class CreateAndDestroy.
3 // Member functions defined in CreateAndDestroy.cpp.
4 #include <string>
5 using std::string;
6
7 #ifndef CREATE_H
8 #define CREATE_H
9
10 class CreateAndDestroy
11 {
12 public:
13     CreateAndDestroy( int, string ); // constructor
14     ~CreateAndDestroy(); // destructor
15 private:
16     int objectID; // ID number for object
17     string message; // message describing object
18 }; // end class CreateAndDestroy
19
20 #endif
```

Prototype for destructor

# 생성자와 소멸자의 호출 시기 예제 (CreatAndDestroy.cpp)

```
1 // Fig. 9.12: CreateAndDestroy.cpp
2 // Member-function definitions for class CreateAndDestroy.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 #include "CreateAndDestroy.h"// include CreateAndDestroy class definition
8
9 // constructor
10 CreateAndDestroy::CreateAndDestroy( int ID, string messageString )
11 {
12     objectID = ID; // set object's ID number
13     message = messageString; // set object's descriptive message
14
15     cout << "Object " << objectID << " constructor runs "
16         << message << endl;
17 } // end CreateAndDestroy constructor
18
19 // destructor
20 CreateAndDestroy::~~CreateAndDestroy()
21 {
22     // output newline for certain objects; helps readability
23     cout << ( objectID == 1 || objectID == 6 ? "\n" : "" );
24
25     cout << "Object " << objectID << " destructor runs "
26         << message << endl;
27 } // end ~CreateAndDestroy destructor
```

Defining the class's destructor

# 생성자와 소멸자의 호출 시기 예제 (driver)

```
1 // Fig. 9.13: fig09_13.cpp
2 // Demonstrating the order in which constructors and
3 // destructors are called.
4 #include <iostream>
5 using std::cout;
6 using std::endl;
7
8 #include "CreateAndDestroy.h" // include CreateAndDestroy class definition
9
10 void create( void ); // prototype
11
12 CreateAndDestroy first( 1, "(global before main)" ); // global object
13
14 int main()
15 {
16     cout << "\nMAIN FUNCTION: EXECUTION BEGINS" << endl;
17     CreateAndDestroy second( 2, "(local automatic in main)" );
18     static CreateAndDestroy third( 3, "(local static in main)" );
19
20     create(); // call function to create objects
21
22     cout << "\nMAIN FUNCTION: EXECUTION RESUMES" << endl;
23     CreateAndDestroy fourth( 4, "(local automatic in main)" );
24     cout << "\nMAIN FUNCTION: EXECUTION ENDS" << endl;
25     return 0;
26 } // end main
```

Object created outside of **main**

Local automatic object created in **main**

Local automatic object created in **main**

# 생성자와 소멸자의 호출 시기 예제 (driver)

```
27
28 // function to create objects
29 void create( void )
30 {
31     cout << "\nCREATE FUNCTION: EXECUTION BEGINS" << endl;
32     CreateAndDestroy fifth( 5, "(local automatic in create)" );
33     static CreateAndDestroy sixth( 6, "(local static in create)" );
34     CreateAndDestroy seventh( 7, "(local automatic in create)" );
35     cout << "\nCREATE FUNCTION: EXECUTION ENDS" << endl;
36 } // end function create
```

Local automatic object created in **create**

Local automatic object created in **create**

# 생성자와 소멸자의 호출 시기 예제 (실행 결과)

object 1    constructor runs    (global before main)

MAIN FUNCTION: EXECUTION BEGINS

object 2    constructor runs    (local automatic in main)

object 3    constructor runs    (local static in main)

CREATE FUNCTION: EXECUTION BEGINS

object 5    constructor runs    (local automatic in create)

object 6    constructor runs    (local static in create)

object 7    constructor runs    (local automatic in create)

CREATE FUNCTION: EXECUTION ENDS

object 7    destructor runs    (local automatic in create)

object 5    destructor runs    (local automatic in create)

MAIN FUNCTION: EXECUTION RESUMES

object 4    constructor runs    (local automatic in main)

MAIN FUNCTION: EXECUTION ENDS

object 4    destructor runs    (local automatic in main)

object 2    destructor runs    (local automatic in main)

object 6    destructor runs    (local static in create)

object 3    destructor runs    (local static in main)

object 1    destructor runs    (global before main)



### 3. Default Memberwise Assignment



# 디폴트 멤버별 (memberwise) 대입

## ☞ 대입 연산자 (=)

- ✓ 해당 객체를 동일한 다른 객체에 대입할 때 사용

- 대입 연산자의 오른쪽에 있는 각 데이터 멤버는 개별적으로 대입 연산자 왼쪽에 있는 데이터 멤버에 1:1 대입 된다.

- ✓ 멤버별 대입은 동적으로 할당된 포인터를 포함하는 데이터 멤버에 대해서는 심각한 문제를 유발 할 수 있으므로 주의

- 같은 메모리를 나중에 두 번 해제할 수 있음

# 디폴트 멤버별 대입 예제 (Date.h)

```
1 // Fig. 9.17: Date.h
2 // Declaration of class Date.
3 // Member functions are defined in Date.cpp
4
5 // prevent multiple inclusions of header file
6 #ifndef DATE_H
7 #define DATE_H
8
9 // class Date definition
10 class Date
11 {
12 public:
13     Date( int = 1, int = 1, int = 2000 ); // default constructor
14     void print();
15 private:
16     int month;
17     int day;
18     int year;
19 }; // end class Date
20
21 #endif
```



# 디폴트 멤버별 대입 예제 (Date.cpp)

```
1 // Fig. 9.18: Date.cpp
2 // Member-function definitions for class Date.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 #include "Date.h" // include definition of class Date from Date.h
8
9 // Date constructor (should do range checking)
10 Date::Date( int m, int d, int y )
11 {
12     month = m;
13     day = d;
14     year = y;
15 } // end constructor Date
16
17 // print Date in the format mm/dd/yyyy
18 void Date::print()
19 {
20     cout << month << '/' << day << '/' << year;
21 } // end function print
```

# 디폴트 멤버별 대입 예제 (driver)

```
1 // Fig. 9.19: fig09_19.cpp
2 // Demonstrating that class objects can be assigned
3 // to each other using default memberwise assignment.
4 #include <iostream>
5 using std::cout;
6 using std::endl;
7
8 #include "Date.h" // include definition of class Date from Date.h
9
10 int main()
11 {
12     Date date1( 7, 4, 2004 );
13     Date date2; // date2 defaults to 1/1/2000
14
15     cout << "date1 = ";
16     date1.print();
17     cout << "\ndate2 = ";
18     date2.print();
19
20     date2 = date1; // default memberwise assignment
21
22     cout << "\n\nAfter default memberwise assignment, date2 = ";
23     date2.print();
24     cout << endl;
25     return 0;
26 } // end main
```

Memberwise assignment assigns data members of **date1** to **date2**

**date2** now stores the same date as **date1**

date1 = 7/4/2004  
date2 = 1/1/2000

After default memberwise assignment, date2 = 7/4/2004



## 4. Copy Constructor



## 복사 생성자 (copy constructor)

```
Date date1( 7, 10, 2013 );
```

```
Date date2(date1);
```

### ✓ 객체 생성시 기존 객체의 값에 의한 전달

- 이미 존재하는 객체의 값을 그대로 이용하여 새로운 객체를 생성한다.

### ✓ 컴파일러는 디폴트 복사 생성자를 제공한다.

- 원래 객체의 각 데이터 멤버를 새로운 객체의 대응되는 데이터 멤버로 복사한다.(즉, 멤버별 대입을 수행)

### ✓ 동적 할당된 포인터를 포함한 데이터 멤버에 대해서는 역시 심각한 문제를 유발 할 수 있다.

# 객체지향프로그래밍 II



## Lecture 3

### 제10장 Class의 이해 및 활용 (Part 1)

#### 1. const Objects and const Member Functions





# 1. `const` Objects and `const` Member Functions



## 상수 객체 (constant objects)

### 최소 특권 원리 (Principle of least privilege)

- ✓ 각 소프트웨어 모듈(객체, 함수, ...)에는 그들에게 필요한 최소한의 권한만을 부여한다.
- ✓ 소프트웨어 설계의 기본적인 원칙 중 하나
- ✓ 객체지향 설계에도 적용됨

### 상수 객체 (constant objects)

- ✓ **const** 키워드를 이용
- ✓ 객체를 수정 (즉 데이터 멤버를 수정) 할 수 없음을 의미
  - 수정을 시도하면 컴파일 오류 발생



## const 멤버 함수

- ✓ 상수 객체는 **const** 멤버 함수만을 호출할 수 있음
- ✓ **const**로 선언된 멤버 함수는 데이터 멤버를 수정할 수 없음
- ✓ 멤버 함수 선언과 정의에 모두 **const** 형으로 지정해야 함
- ✓ 생성자와 소멸자는 **const** 형으로 지정할 수 없음



# const 멤버함수 예제 (time.h)

```
1 // Fig. 10.1: Time.h
2 // Definition of class Time.
3 // Member functions defined in Time.cpp.
4 #ifndef TIME_H
5 #define TIME_H
6
7 class Time
8 {
9 public:
10     Time( int = 0, int = 0, int = 0 ); // default constructor
11
12     // set functions
13     void setTime( int, int, int ); // set time
14     void setHour( int ); // set hour
15     void setMinute( int ); // set minute
16     void setSecond( int ); // set second
17
18     // get functions (normally declared const)
19     int getHour() const; // return hour
20     int getMinute() const; // return minute
21     int getSecond() const; // return second
```

**const** keyword to indicate that member function cannot modify the object

## const 멤버함수 예제 (time.h)

```
22
23 // print functions (normally declared const)
24 void printUniversal() const; // print universal time
25 void printStandard(); // print standard time (should be const)
26 private:
27 int hour; // 0 - 23 (24-hour clock format)
28 int minute; // 0 - 59
29 int second; // 0 - 59
30 }; // end class Time
31
32 #endif
```

# const 멤버함수 예제 (time.cpp)

```
1 // Fig. 10.2: Time.cpp
2 // Member-function definitions for class Time.
3 #include <iostream>
4 using std::cout;
5
6 #include <iomanip>
7 using std::setfill;
8 using std::setw;
9
10 #include "Time.h" // include definition of class Time
11
12 // constructor function to initialize private data;
13 // calls member function setTime to set variables;
14 // default values are 0 (see class definition)
15 Time::Time( int hour, int minute, int second )
16 {
17     setTime( hour, minute, second );
18 } // end Time constructor
19
20 // set hour, minute and second values
21 void Time::setTime( int hour, int minute, int second )
22 {
23     setHour( hour );
24     setMinute( minute );
25     setSecond( second );
26 } // end function setTime
```

# const 멤버함수 예제 (time.cpp)

```
27
28 // set hour value
29 void Time::setHour( int h )
30 {
31     hour = ( h >= 0 && h < 24 ) ? h : 0; // validate hour
32 } // end function setHour
33
34 // set minute value
35 void Time::setMinute( int m )
36 {
37     minute = ( m >= 0 && m < 60 ) ? m : 0; // validate minute
38 } // end function setMinute
39
40 // set second value
41 void Time::setSecond( int s )
42 {
43     second = ( s >= 0 && s < 60 ) ? s : 0; // validate second
44 } // end function setSecond
45
46 // return hour value
47 int Time::getHour() const // get functions should be const
48 {
49     return hour;
50 } // end function getHour
```

const keyword in function definition,  
as well as in function prototype

# const 멤버함수 예제 (time.cpp)

```
52 // return minute value
53 int Time::getMinute() const
54 {
55     return minute;
56 } // end function getMinute
57
58 // return second value
59 int Time::getSecond() const
60 {
61     return second;
62 } // end function getSecond
63
64 // print Time in universal-time format (HH:MM:SS)
65 void Time::printUniversal() const
66 {
67     cout << setfill( '0' ) << setw( 2 ) << hour << ":"
68         << setw( 2 ) << minute << ":" << setw( 2 ) << second;
69 } // end function printUniversal
70
71 // print Time in standard-time format (HH:MM:SS AM or PM)
72 void Time::printStandard() // note lack of const declaration
73 {
74     cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
75         << ":" << setfill( '0' ) << setw( 2 ) << minute
76         << ":" << setw( 2 ) << second << ( hour < 12 ? " AM" : " PM" );
77 } // end function printStandard
```

# const 멤버함수 예제 (driver)

```
1 // Fig. 10.3: fig10_03.cpp
2 // Attempting to access a const object with non-const member functions.
3 #include "Time.h" // include Time class definition
4
5 int main()
6 {
7     Time wakeUp( 6, 45, 0 ); // non-constant object
8     const Time noon( 12, 0, 0 ); // constant object
9
10         // OBJECT      MEMBER FUNCTION
11     wakeUp.setHour( 18 ); // non-const non-const
12
13     noon.setHour( 12 ); // const non-const
14
15     wakeUp.getHour(); // non-const const
16
17     noon.getMinute(); // const const
18     noon.printUniversal(); // const const
19
20     noon.printStandard(); // const non-const
21     return 0;
22 } // end main
```

Cannot invoke non-**const** member functions on a **const** object

*Microsoft Visual C++ .NET compiler error messages:*

```
c:\cpphttp5_examples\ch10\Fig10_01_03\fig10_03.cpp(13) : error C2662:
'Time::setHour' : cannot convert 'this' pointer from 'const Time' to
'Time &' conversion loses qualifiers
c:\cpphttp5_examples\ch10\Fig10_01_03\fig10_03.cpp(20) : error C2662:
'Time::printStandard' : cannot convert 'this' pointer from 'const Time' to
'Time &' conversion loses qualifiers
```



# Member Initializer

## ☞ 멤버 초기화기 (Member initializer)

- ✓ 모든 데이터 멤버는 `member initializer`를 이용하여 초기화 가능
- ✓ `const`형 또는 참조형 데이터 멤버는 반드시 `member initializer`를 이용하여 초기화해야 함

## ☞ 멤버 초기화기 목록 (Member initializer list)

- ✓ 클래스 생성자의 인자 리스트와 함수 시작의 `{` 사이에 위치
- ✓ 복수의 데이터 멤버를 초기화 할 수 있음
- ✓ 클래스 생성자가 수행되기 직전에 수행
- ✓ 구체적인 서식은 다음 예제 참조

# const 멤버의 초기화 예제 (increment.h)

```
1 // Fig. 10.4: Increment.h
2 // Definition of class Increment.
3 #ifndef INCREMENT_H
4 #define INCREMENT_H
5
6 class Increment
7 {
8 public:
9     Increment( int c = 0, int i = 1 ); // default constructor
10
11     // function addIncrement definition
12     void addIncrement()
13     {
14         count += increment;
15     } // end function addIncrement
16
17     void print() const; // prints count and increment
18 private:
19     int count;
20     const int increment; // const data member
21 }; // end class Increment
22
23 #endif
```

Member function declared **const** to prevent errors in situations where an **Increment** object is treated as a **const** object

**const** data member that must be initialized using a member initializer



# const 멤버의 올바른 초기화 (increment.cpp)

```
1 // Fig. 10.5: Increment.cpp
2 // Member-function definitions for class Increment demonstrate using a
3 // member initializer to initialize a constant of a built-in data type.
4 #include <iostream>
5 using std::cout;
6 using std::endl;
7
8 #include "Increment.h" // i
9
10 // constructor
11 Increment::Increment( int c, int i )
12     : count( c ), // initializer for non-const member
13     increment( i ) // required initializer for const member
14 {
15     // empty body
16 } // end constructor Increment
17
18 // print count and increment values
19 void Increment::print() const
20 {
21     cout << "count = " << count << ", increment = " << increment << endl;
22 } // end function print
```

Colon (:) marks the start of a member initializer list

Member initializer for non-**const** member **count**

Required member initializer for **const** member **increment**

# const 멤버의 잘못된 초기화 (increment.cpp)

```
1 // Fig. 10.8: Increment.cpp
2 // Attempting to initialize a constant of
3 // a built-in data type with an assignment.
4 #include <iostream>
5 using std::cout;
6 using std::endl;
7
8 #include "Increment.h" // include definition of class Increment
9
10 // constructor; constant member 'increment' is not initialized
11 Increment::Increment( int c, int i )
12 {
13     count = c; // allowed because count is not constant
14     increment = i; // ERROR: Cannot modify a const object
15 } // end constructor Increment
16
17 // print count and increment values
18 void Increment::print() const
19 {
20     cout << "count = " << count << ", increment = " << increment << endl;
21 } // end function print
```

It is an error to modify a **const** data member; data member **increment** must be initialized with a member initializer

```
C:\cpphttp5_examples\ch10\Fig10_07_09\Increment.cpp(12) : error C2758:
'Increment::increment' : must be initialized in constructor
base/member initializer list
    C:\cpphttp5_examples\ch10\Fig10_07_09\Increment.h(20) :
        see declaration of 'Increment::increment'
C:\cpphttp5_examples\ch10\Fig10_07_09\Increment.cpp(14) : error C2166:
l-value specifies const object
```

# const 멤버의 초기화 예제 (driver)

```
1 // Fig. 10.6: fig10_06.cpp
2 // Program to test class Increment.
3 #include <iostream>
4 using std::cout;
5
6 #include "Increment.h" // include definition of class Increment
7
8 int main()
9 {
10     Increment value( 10, 5 );
11
12     cout << "Before incrementing: ";
13     value.print();
14
15     for ( int j = 1; j <= 3; j++ )
16     {
17         value.addIncrement();
18         cout << "After increment " << j << ": ";
19         value.print();
20     } // end for
21
22     return 0;
23 } // end main
```

```
Before incrementing: count = 10, increment = 5
After increment 1: count = 15, increment = 5
After increment 2: count = 20, increment = 5
After increment 3: count = 25, increment = 5
```

## Ex 9-7: Enhancing class time

**9.7 (*Enhancing Class Time*)** Modify the Time class (Figs. 9.8–9.9) to include a tick member function that increments the time stored in a Time object by one second. The Time object should always remain in a consistent state. Write a program that tests the tick member function in a loop that prints the time in standard format during each iteration of the loop to illustrate that the tick member function works correctly. Be sure to test the following cases:

- a) Incrementing into the next minute.
- b) Incrementing into the next hour.
- c) Incrementing into the next day (i.e., 11:59:59 PM to 12:00:00 AM).

## HW #4 : Class의 이해 및 활용

- 예제 9-7의 기능을 완성하고 보고서 작성하시오
- Deadline : 다음 실습시간 전까지