

Chapter 7. 생성자와 툴

박 종 혁 교수

UCS Lab

(<http://www.parkjonghyuk.net>)

Tel: 970-6702

Email: jhpark1@snut.ac.kr

Learning Objectives

- 생성자
 - 정의
 - 호출
- 도구들
 - `const` 매개변수 수정자
 - 인라인 함수
 - `static` 멤버 데이터

생성자 (Constructor)

- 객체의 초기화
 - 일부 또는 모든 멤버변수 초기화
 - 다른 동작도 가능
- 특별한 멤버함수
 - 객체 선언시 자동으로 호출됨
- 매우 유용한 도구
 - OOP의 핵심 규칙

생성자 정의

- 다른 멤버함수와 동일하게 정의
 - 예외:
 1. 클래스와 동일한 이름을 가져야 함
 2. 값을 리턴 하지 못함; void형도 아님!

생성자 정의의 예

- 생성자를 가지는 클래스 정의:
 - class DayOfYear
{
 public:
 DayOfYear(int monthValue, int dayValue);
 //생성자는 month와 day를 초기화
 void input();
 void output();
 ...
 private:
 int month;
 int day;
}

생성자의 특징

- 생성자의 이름: DayOfYear
 - 클래스의 이름과 동일!
- 생성자 선언부에 리턴 형이 없음
 - void 형도 아님!
- 생성자는 public 섹션에 있음
 - 생성자는 객체가 선언될 때 호출됨
 - private 섹션에 있다면, 객체를 생성할 수 없음!

생성자 호출

- 객체 선언:
 DayOfYear date1(7, 4),
 date2(5, 5);
- 생성자가 호출되면서 객체가 생성
 - 생성자 호출
 - 소괄호 안의 값은 인자로 생성자에게 전달됨
 - 멤버 변수 month와 day가 초기화됨:
 date1.month → 7 date2.month → 5
 date1.dat → 4 date2.day → 5

생성자 등가성 (Constructor Equivalency)

- 다음을 고려해보자:
 - DayOfYear date1, date2
date1.DayOfYear(7, 4); // 허용하지 않음!
date2.DayOfYear(5, 5); // 허용하지 않음!
- 생성자는 다른 멤버 함수와 같은 방법으로 호출할 수 없음!

생성자 코드

- 다른 멤버 함수와 동일하게 생성자도 정의함:

```
DayOfYear::DayOfYear(int monthValue, int  
dayValue)
```

```
{  
    month = monthValue;  
    day = dayValue;  
}
```

- 클래스명::생성자명
- 리턴 형이 없음

선택적 정의

- 이전 정의와 동일한 의미:

```
DayOfYear::DayOfYear(           int monthValue,  
                               int dayValue)  
    : month(monthValue), day(dayValue) ←  
    {...}
```

- 3라인을 초기화 섹션(Initialization Section)이라 함
- 바디 부분은 비어있음
- 향상된 정의 버전

생성자의 추가 목적

- 단지 데이터의 초기화를 위해서는 아님
- 바디 부분을 비어둘 필요는 없음
 - 다른 액션도 추가가능
- 예: 데이터의 검증!
 - `private` 멤버변수에 적절한 데이터가 할당되도록 함
 - 강력한 OOP 규칙

생성자 오버로드

- 다른 함수와 동일하게 생성자도 오버로드가 가능
- 시그니처의 구성요소:
 - 함수의 이름
 - 매개변수 리스트
- 모든 가능한 매개변수 리스트 경우에 대하여 생성자를 제공

디스플레이 7.1 생성자를 정의하는 클래스 (1/3)

Display 7.1 Class with Constructors

```
1  #include <iostream>
2  #include <cstdlib> //for exit
3  using namespace std;

4  class DayOfYear
5  {
6  public:
7      DayOfYear(int monthValue, int dayValue);
8          //Initializes the month and day to arguments.

9      DayOfYear(int monthValue);
10         //Initializes the date to the first of the given month.

11     DayOfYear( ); ← default constructor
12         //Initializes the date to January 1.

13     void input( );
14     void output( );
15     int getMonthNumber( );
16         //Returns 1 for January, 2 for February, etc.
```

디스플레이 7.1 생성자를 정의하는 클래스 (2/3)


```
17     int getDay( );
18 private:
19     int month;
20     int day;
21     void testDate( );
22 };
```

```
23 int main( )
24 {
25     DayOfYear date1(2, 21), date2(5), date3;
26     cout << "Initialized dates:\n";
27     date1.output( ); cout << endl;
28     date2.output( ); cout << endl;
29     date3.output( ); cout << endl;
```


```
30     date1 = DayOfYear(10, 31);
31     cout << "date1 reset to the following:\n";
32     date1.output( ); cout << endl;
33     return 0;
34 }
```

```
35
36 DayOfYear::DayOfYear(int monthValue, int dayValue)
37     : month(monthValue), day(dayValue)
38 {
39     testDate( );
40 }
```

This causes a call to the default constructor. Notice that there are no parentheses.



*an explicit call to the constructor
DayOfYear::DayOfYear*



디스플레이 7.1 생성자를 정의하는 클래스 (3/3)

Display 7.1 Class with Constructors

```
41 DayOfYear::DayOfYear(int monthValue) : month(monthValue), day(1)
42 {
43     testDate( );
44 }

45 DayOfYear::DayOfYear( ) : month(1), day(1)
46 { /*Body intentionally empty.*/}

47 //uses iostream and cstdlib:
48 void DayOfYear::testDate( )
49 {
50     if ((month < 1) || (month > 12))
51     {
52         cout << "Illegal month value!\n";
53         exit(1);
54     }
55     if ((day < 1) || (day > 31))
56     {
57         cout << "Illegal day value!\n";
58         exit(1);
59     }
60 }
```

<Definitions of the other member functions are the same as in Display 6.4.>

SAMPLE DIALOGUE

Initialized dates:
February 21
May 1
January 1
date1 reset to the following:
October 31

인자가 없는 생성자

- 혼동할 수 있다.
- 인자가 없는 표준 함수:
 - 함수 호출: `callMyFunction()`;
 - 빈 소괄호를 포함
- 초기화 인자가 없는 객체 선언:
 - `DayOfYear date1; // This way!`
 - `DayOfYear date(); // No!`
 - 컴파일러는 함수의 선언 또는 원형으로 판단할 소지가 있음
 - 혼란의 가능성~!!

디폴트 생성자

- 인자가 없는 생성자로 정의
- 하나는 항상 정의해야 함
- 자동생성?
 - Yes & No
 - 생성자의 정의가 하나도 없다면 → Yes
 - 하나 이상의 생성자가 정의 되었다면 → No
- 디폴트 생성자가 없을 경우:
 - 다음과 같은 객체 선언은 불가능
 - `MyClass myObject;`

명시적 생성자 호출

- 객체 선언 후에 생성자의 재호출 가능
 - `holiday = DayOfYear(5, 5);`
- 멤버 변수의 세팅에 편리한 방법
- 표준 멤버 함수와는 다른 방법으로 호출됨

명시적 생성자 호출의 예

- 무명 객체(anonymous object)를 리턴

- In Action:

DayOfYear holiday(7, 4);

- 객체 선언할 때 생성자 호출
- 이제 객체를 재 초기화:

holiday = DayOfYear(5, 5);

- 명시적 생성자 호출
- 새로운 무명 객체 호출
- 현재의 객체가 할당됨

클래스 형 멤버 변수

- 클래스 멤버 변수는 어떠한 형도 가능
 - 다른 클래스의 객체도 포함!
 - 클래스 형
 - 강력한 OOP 규칙
- 생성자에 특별한 표기가 필요
 - 멤버 객체의 생성자를 호출

디스플레이 7.3 클래스 멤버 변수 (1/5)

Display 7.3 A Class Member Variable

```
1  #include <iostream>
2  #include<cstdlib>
3  using namespace std;

4  class DayOfYear
5  {
6  public:
7      DayOfYear(int monthValue, int dayValue);
8      DayOfYear(int monthValue);
9      DayOfYear( );
10     void input( );
11     void output( );
12     int getMonthNumber( );
13     int getDay( );
14 private:
15     int month;
16     int day;
17     void testDate( );
18 };
```

The class DayOfYear is the same as in Display 7.1, but we have repeated all the details you need for this discussion.

디스플레이 7.3 클래스 멤버 변수 (2/5)

```
19 class Holiday
20 {
21 public:
22     Holiday( );//Initializes to January 1 with no parking enforcement
23     Holiday(int month, int day, bool theEnforcement);
24     void output( );
25 private:
26     DayOfYear date;
27     bool parkingEnforcement;//true if enforced
28 };

29 int main( )
30 {
31     Holiday h(2, 14, true);
32     cout << "Testing the class Holiday.\n";
33     h.output( );
34
35     return 0;
36 }

37 Holiday::Holiday( ) : date(1, 1), parkingEnforcement(false)
38 { /*Intentionally empty*/}

39 Holiday::Holiday(int month, int day, bool theEnforcement)
40     : date(month, day), parkingEnforcement(theEnforcement)
41 { /*Intentionally empty*/}
```

member variable of a class type

Invocations of constructors from the class DayOfYear.

(continued)

디스플레이 7.3 클래스 멤버 변수 (3/5)

Display 7.3 A Class Member Variable

```
42 void Holiday::output( )
43 {
44     date.output( );
45     cout << endl;
46     if (parkingEnforcement)
47         cout << "Parking laws will be enforced.\n";
48     else
49         cout << "Parking laws will not be enforced.\n";
50 }

51 DayOfYear::DayOfYear(int monthValue, int dayValue)
52                     : month(monthValue), day(dayValue)
53 {
54     testDate( );
55 }
```

디스플레이 7.3 클래스 멤버 변수 (4/5)

```
56 //uses iostream and cstdlib:
57 void DayOfYear::testDate( )
58 {
59     if ((month < 1) || (month > 12))
60     {
61         cout << "Illegal month value!\n";
62         exit(1);
63     }
64     if ((day < 1) || (day > 31))
65     {
66         cout << "Illegal day value!\n";
67         exit(1);
68     }
69 }
70
71 //Uses iostream:
72 void DayOfYear::output( )
73 {
74     switch (month)
75     {
76     case 1:
77         cout << "January "; break;
78     case 2:
79         cout << "February "; break;
80     case 3:
81         cout << "March "; break;
82         .
83         .
84         .
```

The omitted lines are in Display 6.3, but they are obvious enough that you should not have to look there.

디스플레이 7.3 클래스 멤버 변수 (5/5)

Display 7.3 A Class Member Variable

```
82         case 11:
83             cout << "November "; break;
84         case 12:
85             cout << "December "; break;
86         default:
87             cout << "Error in DayOfYear::output. Contact software vendor.";
88     }

89     cout << day;
90 }
```

SAMPLE DIALOGUE

Testing the class Holiday.
February 14
Parking laws will be enforced.

매개변수 전달 방법

- 매개변수 전달의 효율성
 - Call-by-value
 - 복사본의 생성이 필요 → 오버헤드
 - Call-by-reference
 - 실인자의 위치지정자(Placeholder)
 - 가장 효율적인 방법
 - 기본형에서는 두 방법의 차이가 미세함
 - 클래스 형 → 매우 효율적!!
- Call-by-reference 필요성
 - 클래스 형과 같은 큰 데이터의 전달에 효율적

const 매개변수 수정자

- 큰 데이터 형 (클래스)
 - pass-by-reference가 필요
 - 함수에서 데이터 수정을 하지 않아도 필요 (read-only 함수)
- 매개변수 보호
 - constant 매개변수 사용
 - constant call-by-reference 매개변수
 - 형 앞에 *const* 키워드 명시
 - 매개변수 → read-only
 - 수정을 시도하면 컴파일 에러 발생

const의 사용

- All-or-nothing
- 함수에서 매개변수의 수정이 필요 없으면,
 - const로 해당 매개변수 보호
 - 모든 매개변수 보호
- 매개변수로 사용되는 클래스가 사용하는 멤버 함수도 const를 사용하여 일관성을 보장해야 함

인라인 함수

- non-member functions:
 - 함수 선언과 함수 헤더에 *inline* 키워드 사용
- class member functions:
 - 클래스 정의부에 구현함 → 자동적으로 인라인 함수가 됨
- 매우 짧은 함수의 경우에만 사용
- 호출시에 코드가 삽입됨
 - 오버헤드 제거
 - 효율적이지만, 간단한 함수에서만 사용!

인라인 멤버 함수

- 멤버 함수 정의
 - 일반적으로 (다른 파일에) 분리하여 정의
 - 클래스 정의부내에 정의가능
 - 인라인 함수
- 확인: 매우 짧은 함수에만 사용
- 더욱더 효율적
- 너무 길면 → 실제 효율성이 떨어짐!

정적 멤버 (Static Member)

- 정적 멤버 변수
 - 클래스의 모든 객체는 하나의 정적 멤버 변수를 공유
 - 하나의 객체가 정적 멤버 변수를 변경하면 → 모든 객체는 정적 멤버 변수의 변경을 알게 됨
- 추적(tracking)에 유용
 - 멤버 함수가 몇 번 호출 되었는가?
 - 일정한 시간 동안 몇 개의 객체가 존재하는가?
- 형 앞에 *static* 키워드 사용

정적 함수 (Static Function)

- 멤버 함수도 정적으로 사용가능
 - 함수가 객체의 데이터를 참조하지 않고, 클래스의 멤버로 두기를 원한다면 → 정적 함수로 만듦
- 클래스의 외부에서 호출가능
 - non-class 객체(클래스 이름) :
 - 예) `Server::getTurn();`
 - 클래스의 객체 :
 - 예) `myObject.getTurn();`
- 정적 데이터만 사용가능!

디스플레이 7.6 정적 멤버 (1/4)

Display 7.6 Static Members

```
1  #include <iostream>
2  using namespace std;

3  class Server
4  {
5  public:
6      Server(char letterName);
7      static int getTurn( );
8      void serveOne( );
9      static bool stillOpen( );
10 private:
11     static int turn;
12     static int lastServed;
13     static bool nowOpen;
14     char name;
15 };

16 int Server::turn = 0;
17 int Server::lastServed = 0;
18 bool Server::nowOpen = true;
```

디스플레이 7.6 정적 멤버 (2/4)

```
19  int main( )
20  {
21      Server s1('A'), s2('B');
22      int number, count;
23      do
24      {
25          cout << "How many in your group? ";
26          cin >> number;
27          cout << "Your turns are: ";
28          for (count = 0; count < number; count++)
29              cout << Server::getTurn( ) << ' ';
30          cout << endl;
31          s1.serveOne( );
32          s2.serveOne( );
33      } while (Server::stillOpen( ));
34
35      cout << "Now closing service.\n";
36
37      return 0;
38  }
```

디스플레이 7.6 정적 멤버 (3/4)

Display 7.6 Static Members

```
39  Server::Server(char letterName) : name(letterName)
40  { /*Intentionally empty*/}

41  int Server::getTurn( )
42  {
43      turn++;
44      return turn;
45  }
46  bool Server::stillOpen( )
47  {
48      return nowOpen;
49  }

50  void Server::serveOne( )
51  {
52      if (nowOpen && lastServed < turn)
53      {
54          lastServed++;
55          cout << "Server " << name
56              << " now serving " << lastServed << endl;
57      }
```

← Since `getTurn` is static, only static members can be referenced in here.

디스플레이 7.6 정적 멤버 (4/4)

```
58     if (lastServed >= turn) //Everyone served
59         nowOpen = false;
60 }
```

SAMPLE DIALOGUE

How many in your group? **3**

Your turns are: 1 2 3

Server A now serving 1

Server B now serving 2

How many in your group? **2**

Your turns are: 4 5

Server A now serving 3

Server B now serving 4

How many in your group? **0**

Your turns are:

Server A now serving 5

Now closing service.

요약 (1/2)

- 생성자: 클래스 데이터를 자동으로 초기화
 - 객체가 선언 되었을 때 호출
 - 생성자는 클래스와 같은 이름을 가짐
- 디폴트 생성자는 ()가 없음
 - 항상 정의되어야 함
- 클래스 멤버 변수
 - 다른 클래스의 객체를 멤버 변수로 사용 가능
 - 초기화 섹션(initialization-section)이 필요

요약 (2/2)

- Constant call-by-reference 매개변수
 - call-by-value 보다 효율적임
- 인라인은 아주 짧은 함수의 정의에 사용 가능
 - 효율성 향상
- 정적 멤버 변수
 - 클래스의 모든 객체가 공유

Q&A