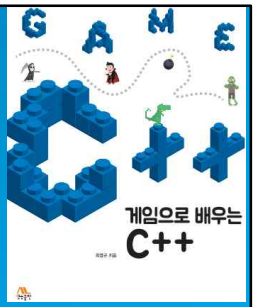
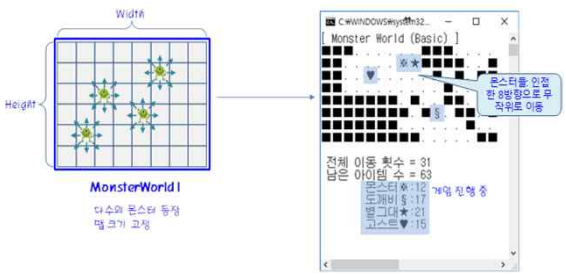
게임으로 배우는 C++



CHAPTER 07

객체의 생성과 소멸

몬스터 월드1(어지러운 세상)



- Monster World
  - 평면에서 몬스터들이 무작위로 움직이면서 세상을 어지럽힘
  - 몬스터는 클래스로, 세상은 2차원 배열로 구현
  - 배열의 각 칸에는 맨 처음에 아이템이 하나씩 놓임
  - 몬스터는 각기 다른 모양으로 화면에 출력
  - 몬스터들이 움직이면서 아이템을 먹음.
  - 인접한 8 방향의 이웃 칸들 중에 하나를 무작위로 선택
  - 몬스터가 움직일 때 마다 맵의 상태와 각 몬스터의 상태 출력

2

7장 학습 목표

- 생성자와 소멸자의 의미를 이해한다.
- 생성자와 일반 멤버 함수의 유사점과 차이점을 이해한다.
- 다양한 생성자를 활용할 수 있는 능력을 기른다.
- 멤버 초기화 리스트의 용도와 필요한 상황들을 이해한다.
- 복사 생성자의 의미와 복사 생성자가 호출되는 상황을 이해한다.
- 함수의 설계와 복사 생성자의 관계를 이해한다.
- 소멸자가 필요한 상황을 이해한다.
- 몬스터 세상 프로그램의 구조와 클래스들을 이해한다.

3

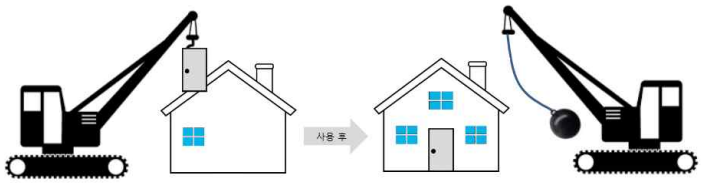
7.1 객체의 생성과 소멸을 도와주는 함수

- 객체의 생성과 파괴?
- 생성자(constructor)란?
- 소멸자(destructor)란?

4

객체의 생성과 파괴?

- 객체의 생성과 파괴




```
void f1()
{
  int x;// 변수 x를 생성
  x=10; // x 초기화
  ...   // x 사용 코드

}       // 변수 x가 파괴됨
```

```
void f2()
{
  Complex c;     // 객체 c를 생성
  c.real = 10;   // 실수부 초기화
  c.imag = 20;   // 허수부 초기화
  ...            // 객체 c를 사용
}                // 객체 c가 파괴됨
```

5

생성자(constructor)란?

- 전통적인 객체 초기화 방법

```
Complex c;      // 객체 c를 생성
c.real = 10;    // 실수부 초기화
c.imag = 20;    // 허수부 초기화
```

```
Complex c;      // 객체 c를 생성
c.set(10,20);   // 실수부와 허수부를 초기화하는 함수 호출
```

- 생성자(constructor)를 사용하면?

```
Complex c(10,20);       // 객체 c를 생성과 동시에 초기화
```

- 생성자 사용 예

```
16행:    ifstream fs(progName);
55행:    answer = string(problem.length(), '-');
```

6

소멸자(destructor)란?

- 소멸자
  - 객체가 소멸될 때 자동으로 호출되는 함수
  - 객체가 소멸될 때 반드시 소멸자가 호출 됨

- 필요한 사례: 게임에서 랭킹을 관리
  - 생성자
    - 게임 객체가 만들어지면 자동으로 저장한 파일에서 데이터를 읽어 현재 순위를 설정
  - 소멸자
    - 게임이 끝나고 게임 객체가 소멸되기 전에 현재 순위를 랭킹 파일에 저장
    - 그래야 다음 게임에서 현재까지의 갱신된 순위가 반영됨

7

7.2 생성자

- 생성자와 일반 함수의 차이점
- 생성자와 일반 함수의 공통점

8

생성자와 일반 함수의 차이점

- 생성자는 반환형이 없고 함수의 이름이 클래스와 동일.
- 생성자는 객체가 생성될 때 오직 한 번만 호출
- 생성자가 하나도 없으면 컴파일러가 기본 생성자를 제공
- 대부분의 경우 public
- 멤버 초기화 리스트를 사용할 수 있음.

```
class Complex {
  double real, imag;
  ...
  Complex( ) { real=imag=0.0; }  // 기본 생성자
  Complex(double r, double i){   // 매개 변수가 있는 생성자
        real = r;                // r을 멤버 변수 real에 복사
        imag = i;                // i를 멤버 변수 imag에 복사
  }
};
```

9

생성자와 일반 함수의 공통점

- 멤버 변수나 멤버 함수를 사용할 수 있음
- 함수 중복(function overloading)이 가능
- 디폴트 매개변수(default function param) 선언 가능

```
class Complex {          // 복소수 클래스 정의
  double real, imag;     // 멤버 변수. 실수부와 허수부
  ...
  Complex(double r=0, double i=0){      // 디폴트 생성자 겸용 생성자
    real = r;            // r을 멤버 변수 real에 복사
    imag = i;            // i를 멤버 변수 imag에 복사
  }
};
```

10

7.3 멤버 초기화 리스트란?

- 멤버 초기화 리스트란?
- 상수 멤버의 초기화
- 참조자 멤버의 초기화
- 멤버 변수가 다른 클래스의 객체인 경우

멤버 초기화 리스트란?

```
class Complex {          // 복소수 클래스 정의
  double real, imag;     // 멤버 변수. 실수부와 허수부
  ...
  // 멤버 초기화 리스트를 이용한 멤버의 초기화
   Complex(double r=0, double i=0) : real(r), imag(i) { }
};
```

- 멤버 초기화 리스트를 반드시 사용해야 하는 상황
  - 상수 멤버의 초기화 (C++11이전 버전)
  - 참조자 멤버의 초기화
  - 멤버 변수가 다른 클래스의 객체인 경우
  - 부모 클래스의 생성자를 골라서 초기화하는 경우

상수 멤버의 초기화

- C++11이전 버전에서 다음 코드는 오류

```
class Complex {          // 복소수 클래스 정의
  double real, imag;     // 멤버 변수. 실수부와 허수부
  const int version = 1; // VS2010에서 오류. 2013에서 OK
  ...
};
```

- 멤버 초기화 리스트를 사용해야 함

```
class Complex {          // 복소수 클래스 정의
  double real, imag;     // 멤버 변수. 실수부와 허수부
  const int version;     // OK
  ...
};
Complex(double r=0, double i=0):real(r),imag(i),version(1) { }
```

13

참조자 멤버의 초기화

- C++11이전 버전에서 다음 코드는 오류

```
class Complex {
  double real, imag;
  double& im = imag;     // VS2010에서 오류. 2013에서 OK
  ...
};
```

- 멤버 초기화 리스트를 사용해야 함

```
class Complex {
  ...;
  int& ref;              // 참조형 멤버
  ...
  Complex(int val) : ref(val) { }
};
```

14

멤버 변수가 다른 클래스의 객체인 경우

```
class Point {    // 화면상의 점을 표현하는 클래스
  int    x, y;    //  점의 x와 y 좌표 값
public:
  Point(int xx = 0, int yy = 0) : x(xx), y(yy) { }
  void setX(int xx) { x = xx; }
  void setY(int yy) { y = yy; }
};
class Line {
  Point p1, p2;
public:
  Line(int x1 = 0, int y1 = 0, int x2 = 0, int y2 = 0) {
    p1.setX(x1);
    p1.setY(y1);
    p2.setX(x2);
    p2.setY(y2);
  }
};
```

```
Line(int x1 = 0, int y1 = 0, int x2 = 0, int y2 = 0)
        : p1(x1,y1), p2(x2,y2) { }
```

15

7.4 소멸자

- 소멸자의 특징

16

소멸자의 특징

- 소멸자는 한 가지 뿐이며 함수 중복 불가능.
- 객체가 소멸될 때 항상 소멸자가 호출된다.
- 컴파일러가 디폴트 소멸자 제공
- 동적으로 할당된 멤버나 상속을 사용한 경우 신중
- 그렇지 않은 대부분의 경우는 생략해도 됨.

```
~Point( )  { cout << “Point소멸자 호출\n”; }
```



```
struct Point {
        int x, y;        // 점의 x와 y 좌표 값
        Point(int xx = 0, int yy = 0) : x(xx), y(yy)
                { cout << "점(" << x << "," << y << ") 생성자\n"; }
        ~Point() { cout << "점(" << x << "," << y << ") 소멸자\n"; }
};
class Circle {
        Point center;    // 원의 중심
        int radius;      // 원의 반경
public:
       Circle(int cx=0, int cy=0, int r=0) : center(cx, cy), radius(r)
               { cout << "원(반지름=" << radius << ") 생성자\n";  }
        ~Circle(){ cout << "원(반지름=" << radius << ") 소멸자\n"; }
};
```

```
void main()
{
    Point p(1, 2);
}
```

지역 변수

```
점(1,2) 생성자
점(1,2) 소멸자
```

```
Point p(1, 2);
void main()
{
}
```

전역 변수

7.5 생성자와 소멸자의 호출 순서

- 생성자의 호출 순서
- 소멸자의 호출 순서

19

생성자의 호출 순서

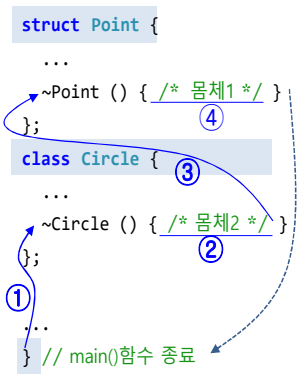
```
struct Point {
  int      x, y;
public:
  Point( int xx = 0, int yy = 0 ): x(xx), y(yy) {/* 몸체1 */}
  ...
};
class Circle {
  Point    center;
  int      radius;
public:
  Circle(int cx=0, int cy=0 , int r=0) : center(cx,cy),radius(r) {/* 몸체2 */}
  ...
};
Circle c (3, 4, 5);

void main() {
    Circle c(3, 4, 5);
    cout << "-----프로그램 종료-----\n";
}
```

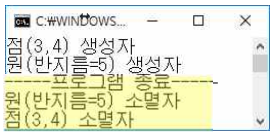
① ② ③ ④

```
C:₩WINDOWS...
점(3,4) 생성자
원(반지름=5) 생성자
-----프로그램 종료-----
원(반지름=5) 소멸자
점(3,4) 소멸자
```

20

소멸자의 호출 순서






```
void main() {
    Circle c(3, 4, 5);
    cout << "-----프로그램 종료-----\n";
}
```

메인 함수 ←

```
Point list[10];
// Point의 기본 생성자가 10번 호출됨
// 함수 종료시 Point의 소멸자도 10번 호출
```


7.6 객체의 복사와 복사 생성자

- 객체의 복사 상황?
- 복사 생성자
- 디폴트 복사 생성자와 대입 연산자

객체의 복사 상황?

- 대입 연산자: b = a;
- 함수에서 객체를 매개변수로 전달하는 경우
- 함수에서 객체를 반환하는 경우

```
Point readPoint() {
        Point p;
        cout << "좌표를 입력해주세요(x,y): ";
        cin >> p.x >> p.y;
        return p;
}
void printPoint(Point p, char* str = "Point") {
        cout << str << "= (" << p.x << "," << p.y << ")\n";
}
void main() {
        Point a;
        a = readPoint();
        printPoint(a, "입력 좌표");
}
```

```
C:₩WINDOWS₩system32₩...
좌표를 입력해주세요(x,y): 1 2
입력 좌표= (1,2)
```


복사 생성자

- 객체의 복사본을 만들기 위해 호출되는 생성자

  클래스명 (const 클래스명 & 참조객체명) { ... }

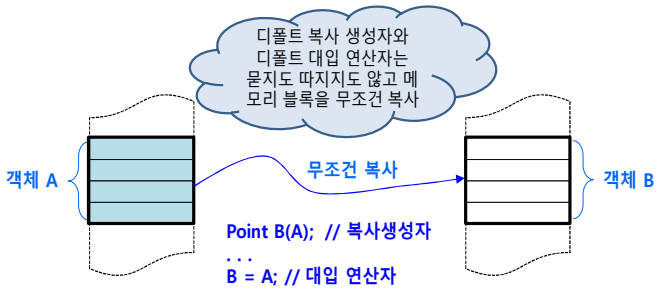
  - 매개변수의 자료형: 반드시 자신과 같은 클래스의 참조형
  - const는 있어도 되고 없어도 됨
  - 생략하면 컴파일러가 알아서 제공함

```
Point( const Point& p): x(p.x), y(p.y) {}
Line( const Line& l): p1(l.p1), p2(l.p2) {}
Circle(const Circle& c): radius(c.radius), center(c.center){}
```

- 호출 상황
  - 다른 객체로부터 새로운 객체 생성: 예) Point p2(p1);
  - 매개변수로 객체가 전달: printPoint()함수
  - 함수가 객체를 반환: readPoint()함수

디폴트 복사 생성자와 대입 연산자

- 컴파일러가 제공하므로 대부분 구현할 필요 없음.
  - 무조건적 복사 방식 (얕은 복사)



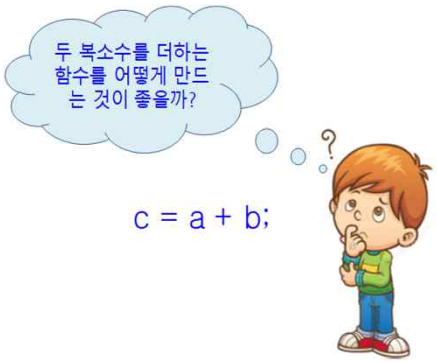
- 깊은 복사가 필요한 경우: 8장
  - 복사 생성자와 대입 연산자를 구현해주어야 함.

7.7 함수의 설계와 객체의 복사

- 함수의 설계와 객체의 복사
- 일반 함수로 구현
- 멤버 함수로 구현
- 객체의 복사와 함수 설계에 대한 제안

함수의 설계와 객체의 복사

- 함수는 다양한 방법으로 구현할 수 있음.
- 예: 두 복소수를 더하는 함수의 구현




```
Complex a, b, c;
add( a, b, c);
add( &a, &b, &c);
c = add( a, b );
c.add( a, b );
c = a.add( b );
. . .
```


일반 함수로 구현

- 결과를 반환하지 않는 방법

함수 원형	사용 방법	복사 생성자	대입 연산자
`void add (Complex, Complex, Complex*);`	`add (a,b,&c);`	2	0
`void add (Complex*, Complex*, Complex*);`	`add (&a,&b,&c);`	0	0
`void add (Complex, Complex, Complex&);`	`add (a,b, c);`	2	0
`void add (Complex&, Complex&, Complex&);`	`add (a,b, c);`	0	0

- 결과를 반환하는 방법

함수 원형	사용 방법	복사 생성자	대입 연산자
`Complex add (Complex, Complex);`	`c = add(a,b);`	3	1
`Complex add (Complex*, Complex*);`	`c = add(&a,&b);`	1	1
`Complex add (Complex&, Complex&);`	`c = add(a,b);`	1	1

멤버 함수로 구현

- 결과를 반환하지 않는 방법

함수 원형 설계 (Complex 클래스의 멤버 함수)	사용 방법	복사 생성자	대입 연산자
`void add (Complex, Complex);`	`c.add(a,b);`	*2*	0
`void add (Complex&, Complex&);`	`c.add(a,b);`	*0*	0

- 결과를 반환하는 방법

함수 원형 설계 (Complex 클래스의 멤버 함수)	사용 방법	복사 생성자	대입 연산자
`Complex add (Complex);`	`c = a.add(b);`	*2*	**1**
`Complex add (Complex&);`	`c = a.add(b);`	*1*	**1**
`Complex operator + (Complex&);`	`c = a + b;`	*1*	**1**

29

객체의 복사와 함수 설계에 대한 제안

- 가능한 한 복사 생성자의 호출을 피하자.
  - `c.add(a,b);`가 `c = a.add(b);` 보다 유리
- 매개 변수로 객체보다는 참조자(Complex&)가 유리
  - 전달된 객체를 수정하지 못하도록 상수형(const)으로 선언
- 연산자 중복을 이용하면 c = a + b; 와 같은 문장 가능
  - 객체의 반환이 필수적. 제한적으로 사용할 것.
- "깊은 복사"가 필요한 경우
  - 개발자가 구현해 주어야 함.
  - 8장에서 공부함.

30

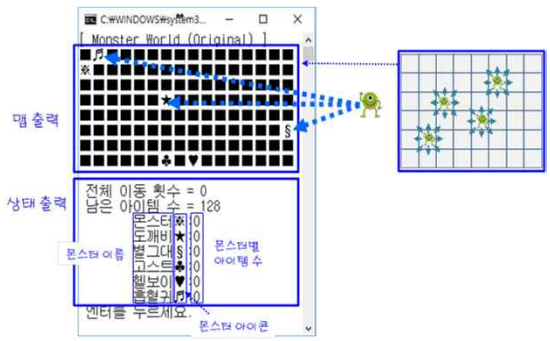
7.8 응용: MonsterWorld

- 몬스터 월드1: 어지러운 세상
- 클래스 설계와 클래스 다이어그램
- 구현
- 고찰

31

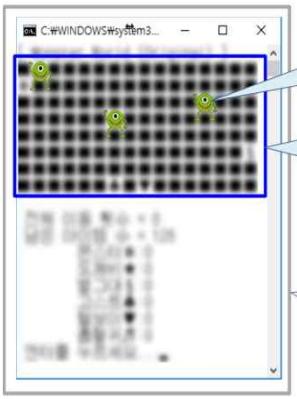
몬스터 월드1(어지러운 세상)

- Monster World
  - 다수의 몬스터 출연
  - 인접한 8 방향의 이웃 칸으로 움직이며 아이템을 먹음
  - 움직일 때 마다 맵의 상태와 각 몬스터의 상태 출력
  - 맵 출력부와 상태 출력부



32

클래스 설계



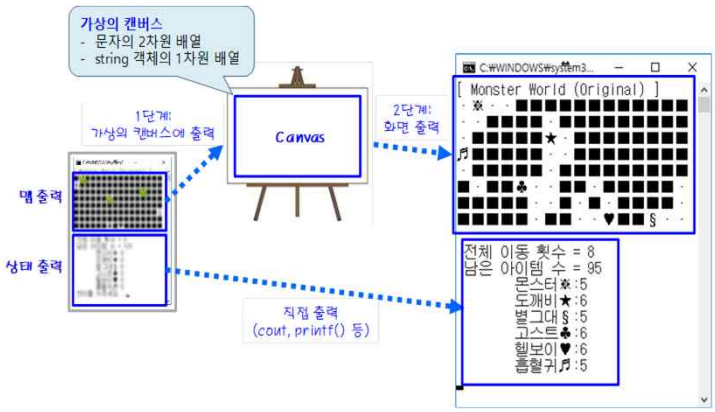
Monster 클래스
- 이름, 아이콘, 현재 위치, 아이템 수
- 움직이고 아이템을 먹는 동작

Canvas 클래스
- 화면 출력용 가상의 캔버스
- 출력할 내용을 캔버스에 먼저 그리고 전체가 그려지면 한꺼번에 실제 화면에 출력

MonsterWorld 클래스
- 전체 몬스터 세상을 관리함
- 캔버스와 몬스터 객체 생성, 처리, 소멸 등을 처리함
- 외부 인터페이스 제공

Canvas 클래스

- 맵 출력을 위한 가상의 캔버스
  - string 클래스를 사용해 구현(각 줄이 하나의 string)



Canvas 클래스 구현

```
class Canvas {
        string line[MAXLINES];   // 화면 출력을 위한 문자열
        int xMax, yMax;          // 맵의 크기
public:
        Canvas(int nx = 10, int ny = 10) : xMax(nx), yMax(ny) {
                for (int y = 0; y < yMax; y++)
                        line[y] = string(xMax * 2, ' ');
        }
        void draw(int x, int y, string val) {
                if (x>=0 && y>=0 && x<xMax && y<yMax)
                        line[y].replace(x * 2, 2, val);
        }
        void clear(string val = " ") {...}
        void print(char *title = "<My Canvas>") {...}
};
```


Monster 클래스

- 몬스터의 속성과 동작을 표현

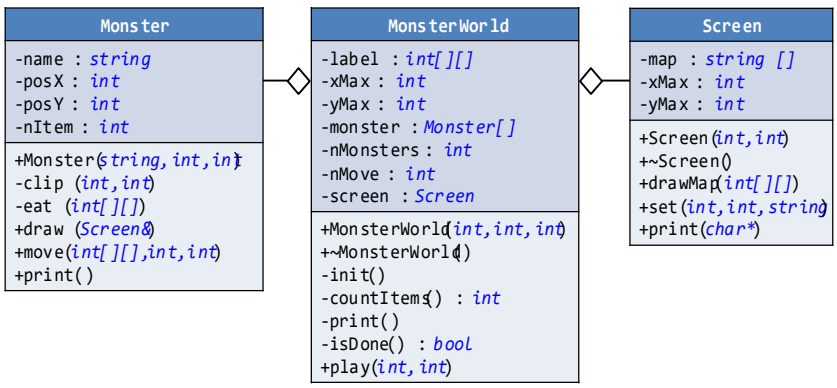
```
class Monster {
    string name, icon;   // 몬스터 이름과 화면 출력용 아이콘
    int x, y, nItem;     // 현재 위치와 먹은 아이템 수
    void clip(int maxx, int maxy) {...}
    void eat(int map[DIM][DIM]) {...}
public:
    Monster(string n="나괴물", string i="※", int px=0, int py=0)
        : name(n), icon(i), x(px), y(py), nItem(0) {}
    ~Monster() { cout << "\t" << name << icon << " 물러갑니다~~~\n"; }
    void draw(Canvas &canvas) { canvas.draw(x, y, icon); }
    void move(int map[DIM][DIM], int maxx, int maxy) {
        switch (rand() % 8) { ... }
        clip(maxx, maxy);
        eat(map);
    }
    void print() { cout<<"\t"<<name << icon << ":" << nItem<<endl;}
};
```

MonsterWorld 클래스

- 전체 게임을 위한 클래스

```
class MonsterWorld {
        int map[DIM][DIM];
        int xMax, yMax, nMon, nMove;
        Monster mon[MAXMONS];
        Canvas  canvas;

        int& Map(int x, int y) { return map[y][x]; }
        bool isDone() { return countItems() == 0; }
        int countItems() {…}
        void print() {…}
public:
        MonsterWorld(int w, int h) : canvas(w, h), xMax(w), yMax(h) {…}
        ~MonsterWorld() { }
        void add( Monster& m) {…}
        void play(int maxwalk, int wait) {…}
};
```


클래스 다이어그램



전체 프로그램

```
#include "MonsterWorld.h"
#include <time.h>
void main()
{
        srand((unsigned int)time(NULL));
        int w = 16, h = 8;
        MonsterWorld game(w, h);
        Monster m("몬스터", "※", rand() % w, rand() % h);
        game.add(m);
        game.add(Monster("도깨비", "§", rand() % w, rand() % h));
        game.add(Monster("별그대", "★", rand() % w, rand() % h));
        game.add(Monster("고스트", "♥", rand() % w, rand() % h));
        game.play(500, 10);
        printf("------게임 종료-------------------\n");
}
```


게임 화면

고찰

- 클래스의 멤버 함수 → 모두 inline 구현
  - 클래스 설계 및 구현. public, private 접근 권한.
  - 생성자와 소멸자. 디폴트 매개변수와 멤버 초기화 리스트.
  - 참조형 매개 변수와 참조자 반환 함수
  - 무명 객체 생성, 디폴트 대입 연산자를 이용한 객체 복사
- 현재는 게임으로 보기 어려움 → 향후 업그레이드
- 개선 방향
  - 맵의 크기를 변경 → 동적 할당(8장)
  - 세상에는 다양한 몬스터가 있다. → 상속(9장)
  - 사용자(게이머) 참여 → 다형성(10장)
  - 좀 더 공평하고 여유 있는 몬스터 → 프렌드+연산자 중복(11장)
  - 랭킹파일 관련 예외상황 처리 →예외처리(12장)
  - 보다 편리한 코딩→ 템플릿(13장)과 STL(14장)

41

7장 요약문제, 연습문제, 실습문제

42

감사합니다!

43