



06

함수 중복과 static 멤버

학습 목표

1. 함수 중복의 개념을 이해하고, 중복 함수를 작성할 수 있다.
2. 디폴트 매개 변수를 이해하고 작성할 수 있다.
3. 함수 중복 시 발생하는 모호성의 경우를 판별할 수 있다.
4. `static` 속성으로 선언된 멤버의 특성을 이해하고, `static` 속성을 활용할 수 있다.

함수 중복

3

□ 함수 중복

▣ 동일한 이름의 함수가 공존

- 다형성
- C 언어에서는 불가능

▣ function overloading

▣ 함수 중복이 가능한 범위

- 보통 함수들 사이
- 클래스의 멤버 함수들 사이
- 상속 관계에 있는 기본 클래스와 파생 클래스의 멤버 함수들 사이

□ 함수 중복 성공 조건

- ▣ 중복된 함수들의 이름 동일
- ▣ 중복된 함수들의 매개 변수 타입이 다르거나 개수가 달라야 함
- ▣ 리턴 타입은 함수 중복과 무관

함수 중복 성공 사례

4

```
int sum(int a, int b, int c) {  
    return a + b + c;  
}  
  
double sum(double a, double b) {  
    return a + b;  
}  
  
int sum(int a, int b) {  
    return a + b;  
}
```

성공적으로 중복된 sum() 함수들

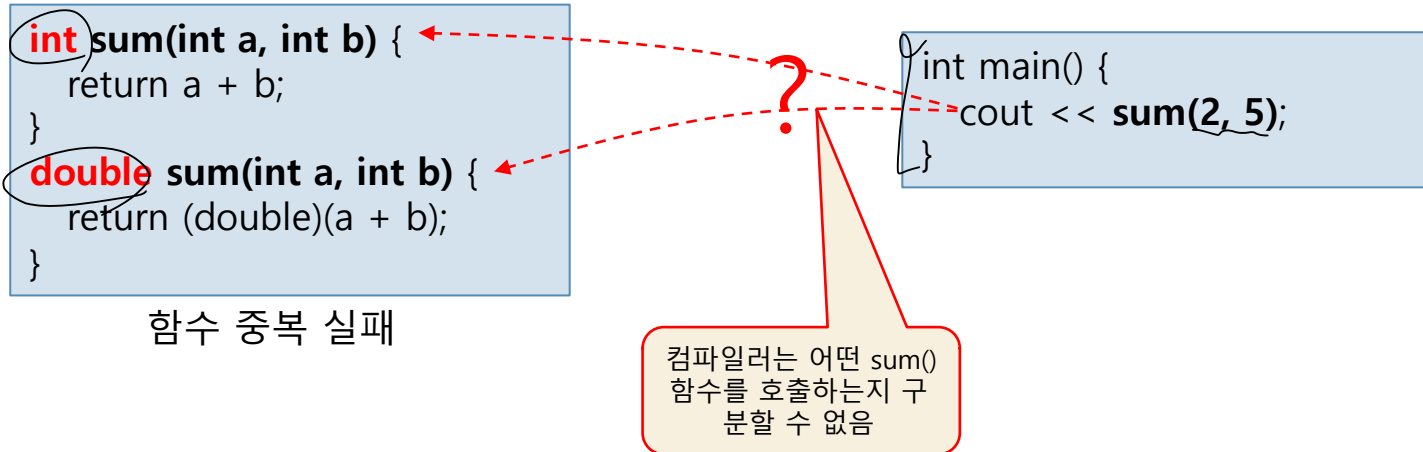
```
int main(){  
    cout << sum(2, 5, 33);  
  
    cout << sum(12.5, 33.6);  
  
    cout << sum(2, 6);  
}
```

중복된 sum() 함수 호출.
컴파일러가 구분

함수 중복 실패 사례

5

- 리턴 타입이 다르다고 함수 중복이 성공하지 않는다.



함수 중복의 편리함

6

- 동일한 이름을 사용하면 함수 이름을 구분하여 기억할 필요 없고, 함수 호출을 잘못하는 실수를 줄일 수 있음

```
void msg1() {  
    cout << "Hello";  
}  
void msg2(string name) {  
    cout << "Hello, " << name;  
}  
void msg3(int id, string name) {  
    cout << "Hello, " << id << " " << name;  
}
```

(a) 함수 중복하지 않는 경우



```
void msg() {  
    cout << "Hello";  
}  
void msg(string name) {  
    cout << "Hello, " << name;  
}  
void msg(int id, string name) {  
    cout << "Hello, " << id << " " << name;  
}
```

(b) 함수 중복한 경우

함수 중복하면 함수 호출의 편리함.
오류 가능성 줄임

예제 6-1 big() 함수 중복 연습

7

큰 수를 리턴하는 다음 두 개의 big 함수를 중복 구현하라.

매개변수 타입이
다름

```
int big(int a, int b);    // a와 b 중 큰 수 리턴
int big(int a[], int size); // 배열 a[]에서 가장 큰 수 리턴
```

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

```
int big(int a, int b) { // a와 b 중 큰 수 리턴
    if(a>b) return a;
    else return b;
}
```

```
int big(int a[], int size) { // 배열 a[]에서 가장 큰 수 리턴
    int res = a[0];
    for(int i=1; i<size; i++)
        if(res < a[i]) res = a[i];
    return res;
}
```

```
int main() {
    int array[5] = {1, 9, -2, 8, 6};
    cout << big(2,3) << endl;
    cout << big(array, 5) << endl;
}
```

예제 6-2(실습) sum() 함수 중복 연습

8

함수 sum()을 호출하는 경우가 다음과 같을 때, 함수 sum()을 중복 구현하라. sum()의 첫 번째 매개 변수는 두 번째 매개 변수보다 작은 정수 값으로 호출된다고 가정한다.

```
sum(3,5); // 3~5까지의 합을 구하여 리턴
sum(3);   // 0~3까지의 합을 구하여 리턴
sum(100); // 0~100까지의 합을 구하여 리턴
```

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

int sum(int a, int b) { // a에서 b까지 합하기
    int s = 0;
    for(int i=a; i<=b; i++)
        s += i;
    return s;
}

int sum(int a) { // 0에서 a까지 합하기
    int s = 0;
    for(int i=0; i<=a; i++)
        s += i;
    return s;
}

int main() {
    cout << sum(3, 5) << endl;
    cout << sum(3) << endl;
    cout << sum(100) << endl;
}
```

12
6
5050

생성자 함수 중복

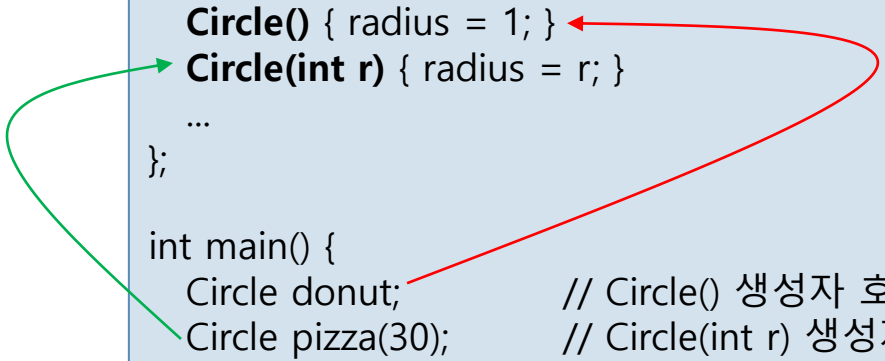
9

□ 생성자 함수 중복 가능

▣ 생성자 함수 중복 목적

- 객체 생성시, 매개 변수를 통해 다양한 형태의 초깃값 전달

```
class Circle {  
    .....  
public:  
    Circle() { radius = 1; }  
    Circle(int r) { radius = r; }  
    ...  
};  
  
int main() {  
    Circle donut;           // Circle() 생성자 호출  
    Circle pizza(30);       // Circle(int r) 생성자 호출  
}
```



string 클래스의 생성자 중복 사례

10

```
class string {  
    ....  
public:  
    string(); // 빈 문자열을 가진 스트링 객체 생성  
    string(string& str); // str을 복사한 새로운 스트링 객체 생성  
    string(char* s); // '₩0'로 끝나는 C-스트링 s를 스트링 객체로 생성  
    ....  
};
```

```
string str; // 빈 문자열을 가진 스트링 객체  
string address("서울시 성북구 삼선동 389");  
string copyAddress(address); // address의 문자열을 복사한 별도의 copyAddress 생성
```

소멸자 함수 중복

11

- 소멸자 함수 중복 불가
 - ▣ 소멸자는 매개 변수를 가지지 않음
 - ▣ 한 클래스 내에서 소멸자는 오직 하나만 존재

디폴트 매개 변수

12

- 디폴트 매개 변수(default parameter)
 - ▣ 매개 변수에 값이 넘어오지 않는 경우,/디폴트 값을 받도록 선언된 매개 변수
 - ‘매개 변수 = 디폴트값’ 형태로 선언

- 디폴트 매개 변수 선언 사례

```
void star(int a=5); // a의 디폴트 값은 5
```

- 디폴트 매개 변수를 가진 함수 호출

```
star(); // 매개 변수 a에 디폴트 값 5가 전달됨. star(5);와 동일  
star(10); // 매개 변수 a에 10을 넘겨줌
```

디폴트 매개 변수 사례

13

□ 사례 1

```
void msg(int id, string text="Hello"); // text의 디폴트 값은 "Hello"
```

```
msg(10); // msg(10, "Hello"); 호출과 동일. id에 10, text에 "Hello" 전달
```

```
msg(20, "Good Morning"); // id에 20, text에 "Good Morning" 전달
```

호출 오류

```
msg(); // 컴파일 오류. 첫 번째 매개 변수 id에 반드시 값을 전달하여야 함
```

```
msg("Hello"); // 컴파일 오류. 첫 번째 매개 변수 id에 값이 전달되지 않았음
```

디폴트 매개 변수에 관한 제약 조건

14

- 디폴트 매개 변수는 보통 매개 변수 앞에 선언될 수 없음
 - 디폴트 매개 변수는 끝 쪽에 몰려 선언되어야 함

컴파일 오류

```
void calc(int a, int b=5, int c, int d=0); // 컴파일 오류  
void sum(int a=0, int b, int c); // 컴파일 오류
```

```
void calc(int a, int b=5, int c=0, int d=0); // 컴파일 성공
```



매개 변수에 값을 정하는 규칙

15

□ 사례 2

```
void square(int width=1, int height=1);
```

디폴트 매개 변수를
가진 square()

```
void square(int width=1, int height=1);
```

square();	→	square(<u> </u> , <u> </u>);	→	square(1 , 1);
square(5);	→	square(5, <u> </u>);	→	square(5, 1);
square(3, 8);	→	square(3, 8);	→	square(3, 8);

컴파일러에 의해
변환되는 과정

디폴트 매개 변수 사례

16

□ 사례 3

```
void g(int a, int b=0, int c=0, int d=0);
```

디폴트 매개 변수를
가진 함수

```
void g(int a, int b=0, int c=0, int d=0);
```

g(10);	→	g(10, <u> </u> , <u> </u> , <u> </u>);	→	g(10, 0, 0, 0);
g(10, 5);	→	g(10, 5 , <u> </u> , <u> </u>);	→	g(10, 5, 0, 0);
g(10, 5, 20);	→	g(10, 5, 20, <u> </u>);	→	g(10, 5, 20, 0);
g(10, 5, 20, 30);	→	g(10, 5, 20, 30);	→	g(10, 5, 20, 30);

컴파일러에 의해
변환되는 과정

예제 6-3 디폴트 매개 변수를 가진 함수 선언 및 호출

17

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

```
// 원형 선언
```

```
void star(int a=5);
void msg(int id, string text="");
```

디폴트
매개 변수 선언

```
// 함수 구현
```

```
void star(int a) {
    for(int i=0; i<a; i++)
        cout << '*';
    cout << endl;
}
```

```
void msg(int id, string text) {
    cout << id << ' ' << text << endl;
}
```

동일한
코드

```
void star(int a=5) {
    for(int i=0; i<a; i++)
        cout << '*';
    cout << endl;
}
```

```
void msg(int id, string text="") {
    cout << id << ' ' << text << endl;
}
```

```
int main() {
    // star() 호출
    star();
    star(10);
```

star(5);

```
    // msg() 호출
    msg(10);
    msg(10, "Hello");
}
```

msg(10, "");

```
*****
*****
10
10 Hello
```

예제 6-4(실습) 디폴트 매개 변수를 가진 함수 만들기 연습

18

함수 f()를 호출하는 경우가 다음과 같을 때 f()를 디폴트 매개 변수를 가진 함수로 작성하라.

빈 칸이 10개 출력됨

```
%%%%%%%%%  
aaaaaaaaa  
aaaaaaaaa  
aaaaaaaaa  
aaaaaaaaa  
aaaaaaaaa  
aaaaaaaaa
```

```
f(); // 한 줄에 빈칸을 10개 출력한다.  
f('%'); // 한 줄에 '%'를 10개 출력한다.  
f('@', 5); // 다섯 줄에 '@'를 10개 출력한다.
```

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

```
// 원형 선언
```

```
void f(char c=' ', int line=1);
```

```
// 함수 구현
```

```
void f(char c, int line) {  
    for(int i=0; i<line; i++) {  
        for(int j=0; j<10; j++)  
            cout << c;  
        cout << endl;  
    }  
}
```

```
int main() {
```

```
    f(); // 한줄에 빈칸을 10개 출력한다.
```

```
    f('%'); // 한 줄에 '%'를 10개 출력한다.
```

```
    f('@', 5); // 5 줄에 '@' 문자를 10개 출력한다.
```

```
}
```

함수 중복 간소화

19

▣ 디폴트 매개 변수의 장점 - 함수 중복 간소화

```
class Circle {  
    .....  
public:  
    Circle() { radius = 1; }  
    Circle(int r) { radius = r; }  
    .....  
};
```

```
class Circle {  
    .....  
public:  
    Circle(int r=1) { radius = r; }  
    .....  
};
```

2 개의 생성자 함수를
디폴트 매개 변수를 가진
하나의 함수로 간소화

▣ 중복 함수들과 디폴트 매개 변수를 가진 함수를 함께 사용 불가

```
class Circle {  
    .....  
public:  
    Circle() { radius = 1; }  
    Circle(int r) { radius = r; }  
    Circle(int r=1) { radius = r; }  
    .....  
};
```

중복된 함수와
동시 사용 불가

예제 6-5(실습) 디폴트 매개 변수를 이용하여 중복 함수 간소화 연습

20

다음 두 개의 중복 함수를 디폴트 매개 변수를 가진 하나의 함수로 작성하라.

```
void fillLine() { // 25 개의 '*' 문자를 한 라인에 출력
    for(int i=0; i<25; i++) cout << '*';
    cout << endl;
}
void fillLine(int n, char c) { // n개의 c 문자를 한 라인에 출력
    for(int i=0; i<n; i++) cout << c;
    cout << endl;
}
```

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

void fillLine(int n=25, char c='*') { // n개의 c 문자를 한 라인에 출력
    for(int i=0; i<n; i++) cout << c;
    cout << endl;
}

int main() {
    fillLine(); // 25개의 '*'를 한 라인에 출력
    fillLine(10, '%'); // 10개의 '%'를 한 라인에 출력
}
```

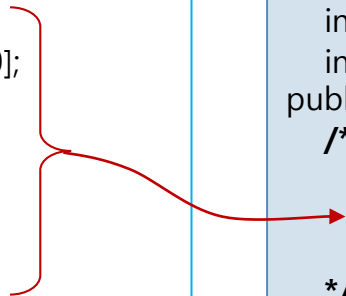
```
*****
%%%%%%%%%
```

예제 6-6(실습) 생성자 함수의 중복 간소화

21

다음 클래스에 중복된 생성자를 디폴트 매개 변수를 가진 하나의 생성자로 작성하라.

```
class MyVector{
    int *p;
    int size;
public:
    MyVector() {
        p = new int [100];
        size = 100;
    }
    MyVector(int n) {
        p = new int [n];
        size = n;
    }
    ~MyVector() { delete [] p; }
};
```



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

```
class MyVector{
    int *p;
    int size;
public:
    /*
```

이곳에 디폴트 매개변수를 가진 생성자 작성하라

```
*/
    ~MyVector() { delete [] p; }
};
```

```
int main() {
    MyVector *v1, *v2;
    v1 = new MyVector(); // 디폴트로 정수 100개의 배열 동적 할당
    v2 = new MyVector(1024); // 정수 1024개의 배열 동적 할당

    delete v1;
    delete v2;
}
```

예제 6-6(실습) 생성자 함수의 중복 간소화(정답)

22

```
class MyVector{
    int *p;
    int size;
public:
    MyVector() {
        p = new int [100];
        size = 100;
    }
    MyVector(int n) {
        p = new int [n];
        size = n;
    }
    ~MyVector() { delete [] p; }
};
```

정답

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

class MyVector{
    int *p;
    int size;
public:
    MyVector(int n=100) {
        p = new int [n];
        size = n;
    }
    ~MyVector() { delete [] p; }
};

int main() {
    MyVector *v1, *v2;
    v1 = new MyVector(); // 디폴트로 정수 100개의 배열 동적 할당
    v2 = new MyVector(1024); // 정수 1024개의 배열 동적 할당

    delete v1;
    delete v2;
}
```

위임생성자로 작성할 수도 있음(3.4절참고)

```
MyVector() : MyVector(100) { }
MyVector(int n) {
    p = new int [n];
    size = n;
}
```

함수 중복의 모호성

23

- 함수 중복이 모호하여 컴파일러가 어떤 함수를 호출하는지 판단하지 못하는 경우
 - ▣ 형 변환으로 인한 모호성
 - ▣ 참조 매개 변수로 인한 모호성
 - ▣ 디폴트 매개 변수로 인한 모호성

형 변환으로 인한 함수 중복의 모호성

24

□ 매개 변수의 형 변환으로 인한 중복 함수 호출의 모호성

```
double square(double a) {  
    return a*a;  
}  
int main() {  
    cout << square(3);  
}
```

int 타입 3이
double 로 자
동 형 변환

(a) 정상 컴파일

```
float square(float a) {  
    return a*a;  
}  
double square(double a) {  
    return a*a;  
}  
int main() {  
    cout << square(3.0);  
    cout << square(3);  
}
```

3.0은 double
타입이므로
모호하지 않음

int 타입 3을
double로 변환
할지 float로 변
환할 지 모호함

(b) 모호한 호출, 컴파일 오류

예제 6-7 형 변환으로 인해 함수 중복이 모호한 경우

25

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

float square(float a) {
    return a*a;
}

double square(double a) {
    return a*a;
}

int main() {
    cout << square(3.0); // square(double a); 호출
    cout << square(3); // 컴파일 오류
}
```

square
오버로드된 함수 "square"의 인스턴스 중 두 개 이상이 인수 목록과 일치합니다.
함수 "square(float a)"
함수 "square(double a)"
인수 형식이 (int) 입니다.

예제 6-8 참조 매개 변수로 인한 함수 중복의 모호성

26

두 함수는
근본적으로
중복 시킬
수 없다.

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

int add(int a, int b) {
    return a + b;
}

int add(int a, int &b) {
    b = b + a;
    return b;
}

int main(){
    int s=10, t=20;
    cout << add(s, t); // 컴파일 오류
}
```

값에 의한 호출

참조 매개 변수

참조에 의한 호출

call by value인지
call by reference인지 모호

예제 6-9 디폴트 매개 변수로 인한 함수 중복의 모호성

27

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

```
void msg(int id) {
    cout << id << endl;
}
```

```
void msg(int id, string s="") {
    cout << id << ":" << s << endl;
}
```

```
int main(){
    msg(5, "Good Morning"); // 정상 컴파일. 두 번째 msg() 호출
    msg(6); // 함수 호출 모호. 컴파일 오류
}
```

디폴트 매개 변수를 이용하고 있는지 모호함

static 멤버와 non-static 멤버

28



사람은 모두 각자의 눈을 가지고 태어난다.



사람이 태어나기 전에 공기가 있으며, 모든 사람은 공기를 공유한다. 공기 역시 각 사람의 것이다.

static 멤버와 non-static 멤버의 특성

29

□ static

▣ 변수와 함수에 대한 기억 부류의 한 종류

- 생명 주기 - 프로그램이 시작될 때 생성, 프로그램 종료 시 소멸
- 사용 범위 - 선언된 범위, 접근 지정에 따름

□ 클래스의 멤버

▣ static 멤버

- 프로그램이 시작할 때 생성
- 클래스 당 하나만 생성, 클래스 멤버라고 불림
- 클래스의 모든 인스턴스(객체)들이 공유하는 멤버

▣ non-static 멤버

- 객체가 생성될 때 함께 생성
- 객체마다 객체 내에 생성
- 인스턴스 멤버라고 불림

static 멤버 선언

30

□ 멤버의 static 선언

```
class Person {  
public:  
    int money; // 개인 소유의 돈  
    void addMoney(int money) {  
        this->money += money;  
    }  
};
```

non-static 멤버 선언

static 멤버 변수 선언

static 멤버 함수 선언

```
static int sharedMoney; // 공금  
static void addShared(int n) {  
    sharedMoney += n;  
}  
};
```

static 변수 공간 할당.
프로그램의 전역 공간에 선언

```
int Person::sharedMoney = 10; // sharedMoney를 10으로 초기화
```

□ static 멤버 변수 생성

- 전역 변수로 생성
- 전체 프로그램 내에 한 번만 생성

□ static 멤버 변수에 대한 외부 선언이 없으면 다음과 같은 링크 오류

주의

컴파일
성공

1>----- 빌드 시작: 프로젝트: StaticSample1, 구성: Debug Win32 -----

1> StaticSample1.cpp

1>StaticSample1.obj : error LNK2001: "public: static int Person::sharedMoney" (?sharedMoney@Person@@@2HA) 외부 기호를 확인할 수 없습니다.

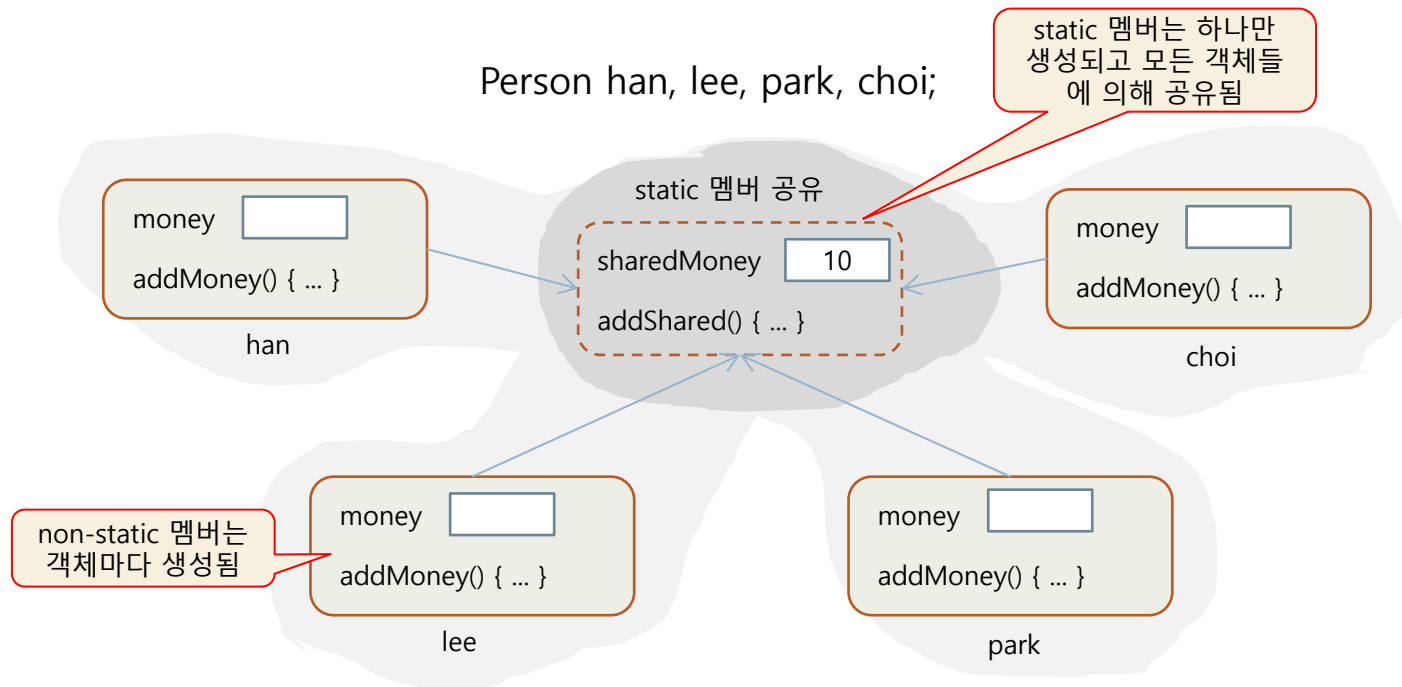
1>C:\WC++\Wchap6\Debug\그림 6-9.exe : fatal error LNK1120: 1개의 확인할 수 없는 외부 참조입니다.

===== 빌드: 성공 0, 실패 1, 최신 0, 생략 0 =====

링크
오류

static 멤버와 non-static 멤버의 관계

31



- han, lee, park, choi 등 4 개의 Person 객체 생성
- sharedMoney와 addShared() 함수는 하나만 생성되고 4 개의 객체들의 의해 공유됨
- sharedMoney와 addShared() 함수는 han, lee, park, choi 객체들의 멤버임

static 멤버와 non-static 멤버 비교

32

항목	non-static 멤버	static 멤버
선언 사례	<pre>class Sample { int n; void f(); };</pre>	<pre>class Sample { <u>static</u> int n; <u>static</u> void f(); };</pre>
공간 특성	멤버는 객체마다 별도 생성 • <u>인스턴스 멤버</u> 라고 부름	멤버는 클래스 당 하나 생성 • 멤버는 객체 내부가 아닌 별도의 공간에 생성 • <u>클래스 멤버</u> 라고 부름
시간적 특성	<u>객체와 생명을 같이 함</u> • 객체 생성 시에 멤버 생성 • 객체 소멸 시 함께 소멸 • 객체 생성 후 객체 사용 가능	<u>프로그램과 생명을 같이 함</u> • 프로그램 시작 시 멤버 생성 • 객체가 생기기 전에 이미 존재 • 객체가 사라져도 여전히 존재 • 프로그램이 종료될 때 함께 소멸
공유의 특성	공유되지 않음 • 멤버는 객체 별로 따로 <u>공간 유지</u>	동일한 클래스의 <u>모든 객체들에 의해 공유됨</u>

static 멤버 사용 : 객체의 멤버로 접근

33

- static 멤버는 객체 이름이나 객체 포인터로 접근
 - ▣ 보통 멤버처럼 접근할 수 있음

객체.static멤버
객체포인터->static멤버

클래스::static 멤버

- ▣ Person 타입의 객체 lee와 포인터 p를 이용하여 static 멤버를 접근하는 예

```
Person lee;  
lee.sharedMoney = 500; // 객체.static멤버 방식
```

```
Person *p;  
p = &lee;  
p->addShared(200); // 객체포인터->static멤버 방식
```

(*p).addShared(200)

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

```
class Person {
public:
    int money; // 개인 소유의 돈
    void addMoney(int money) {
        this->money += money;
    }

    static int sharedMoney; // 공금
    static void addShared(int n) {
        sharedMoney += n;
    }
};
```

```
// static 변수 생성. 전역 공간에 생성
int Person::sharedMoney=10; // 10으로 초기화
```

```
// main() 함수
int main() {
    Person han;
    han.money = 100; // han의 개인 돈=100
    han.sharedMoney = 200; // static 멤버 접근, 공금=200
```

```
    Person lee;
    lee.money = 150; // lee의 개인 돈=150
    lee.addMoney(200); // lee의 개인 돈=350
    lee.addShared(200); // static 멤버 접근, 공금=400
```

```
    cout << han.money << ' '
         << lee.money << endl;
    cout << han.sharedMoney << ' '
         << lee.sharedMoney << endl;
}
```

100 350
400 400

han과 lee의 money는 각각 100, 350

han과 lee의 sharedMoney는 공통 400

main()이 시작하기 직전

sharedMoney 10

addShared() { ... }

sharedMoney ~~10~~ 200

addShared() { ... }

han

money 100

addMoney() { ... }

sharedMoney 200

addShared() { ... }

han

money 100

addMoney() { ... }

lee

money ~~150~~ 350

addMoney() { ... }

Person lee;
lee.money = 150;
lee.addMoney(200);

sharedMoney ~~200~~ 400

addShared() { ... }

han

money 100

addMoney() { ... }

lee

money 350

addMoney() { ... }

lee.addshared(200);

static 멤버 사용 : 클래스명과 범위 지정 연산자 (::)로 접근

35

- 클래스 이름과 범위 지정 연산자(::)로 접근 가능
 - ▣ static 멤버는 클래스마다 오직 한 개만 생성되기 때문

클래스명::static멤버

객체

```
han.sharedMoney = 200;  
lee.addShared(200);
```

<->
<->

직접적으로 접근
↓
클래스

```
Person::sharedMoney = 200;  
Person::addShared(200);
```

- ▣ non-static 멤버는 클래스 이름을 접근 불가

~~Person::money = 100; // 컴파일 오류. non-static 멤버는 클래스 명으로 접근불가~~
~~Person::addMoney(200); // 컴파일 오류. non-static 멤버는 클래스 명으로 접근불가~~

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

```
class Person {
public:
    int money; // 개인 소유의 돈
    void addMoney(int money) {
        this->money += money;
    }
}
```

```
static int sharedMoney; // 공금
static void addShared(int n) {
    sharedMoney += n;
}
};
```

```
// static 변수 생성. 전역 공간에 생성
int Person::sharedMoney=10; // 10으로 초기화
```

```
// main() 함수
int main() {
```

```
    Person::addShared(50); // static 멤버 접근, 공금=60
    cout << Person::sharedMoney << endl;
```

```
    Person han;
    han.money = 100;
    han.sharedMoney = 200; // static 멤버 접근, 공금=200
    Person::sharedMoney = 300; // static 멤버 접근, 공금=300
    Person::addShared(100); // static 멤버 접근, 공금=400
```

```
    cout << han.money << ' '
        << Person::sharedMoney << endl;
}
```

60
100 400 sharedMoney 400

han의 money 100

han 객체가 생기기전부터
static 멤버 접근

main()이 시작하기 직전

sharedMoney 10
addShared() { ... }

Person::addShared(50);

sharedMoney 10 60
addShared() { ... }

Person han;

han

sharedMoney 60
addShared() { ... }

money
addMoney() { ... }

han.money = 100;
han.sharedMoney = 200;

han

sharedMoney 200
addShared() { ... }

money 100
addMoney() { ... }

Person::sharedMoney = 300;
Person::addShared(100);

han

sharedMoney 200 300 400
addShared() { ... }

money 100
addMoney() { ... }

static 활용

37

□ static의 주요 활용

- 전역 변수나 전역 함수를 클래스에 캡슐화
 - 전역 변수나 전역 함수를 가능한 사용하지 않도록
 - 전역 변수나 전역 함수를 static으로 선언하여 클래스 멤버로 선언
- 객체 사이에 공유 변수를 만들고자 할 때
 - static 멤버를 선언하여 모든 객체들이 공유

예제 6-10 static 멤버를 가진 Math 클래스 작성

38

왼쪽 코드를 static 멤버를 가진 Math 클래스로 작성하고 멤버 함수를 호출하라.

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

int abs(int a) { return a>0?a:-a; }
int max(int a, int b) { return a>b?a:b; }
int min(int a, int b) { return (a>b)?b:a; }

int main() {
    cout << abs(-5) << endl;
    cout << max(10, 8) << endl;
    cout << min(-3, -8) << endl;
}
```

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

class Math {
public:
    static int abs(int a) { return a>0?a:-a; }
    static int max(int a, int b) { return (a>b)?a:b; }
    static int min(int a, int b) { return (a>b)?b:a; }
};

int main() {
    cout << Math::abs(-5) << endl;
    cout << Math::max(10, 8) << endl;
    cout << Math::min(-3, -8) << endl;
}
```

5
10
-8

(a) 전역 함수들을 가진 좋지 않음 코딩 사례

(b) Math 클래스를 만들고 전역 함수들을 static 멤버로 캡슐화한 프로그램

예제 6-11 static 멤버를 공유의 목적으로 사용하는 예

생존하고 있는 원의 개수 = 10
생존하고 있는 원의 개수 = 0
생존하고 있는 원의 개수 = 1
생존하고 있는 원의 개수 = 2

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

class Circle {
private:
    static int numOfCircles;
    int radius;
public:
    Circle(int r=1);
    ~Circle() { numOfCircles--; } // 생성된 원의 개수 감소
    double getArea() { return 3.14*radius*radius;}
    static int getNumOfCircles() { return numOfCircles; }
};

Circle::Circle(int r) {
    radius = r;
    numOfCircles++; // 생성된 원의 개수 증가
}

int Circle::numOfCircles = 0; // 0으로 초기화

int main() {
    Circle *p = new Circle[10]; // 10개의 생성자 실행
    cout << "생존하고 있는 원의 개수 = " << Circle::getNumOfCircles() << endl;

    delete [] p; // 10개의 소멸자 실행
    cout << "생존하고 있는 원의 개수 = " << Circle::getNumOfCircles() << endl;

    Circle a; // 생성자 실행
    cout << "생존하고 있는 원의 개수 = " << Circle::getNumOfCircles() << endl;

    Circle b; // 생성자 실행
    cout << "생존하고 있는 원의 개수 = " << Circle::getNumOfCircles() << endl;
}
```

static 멤버 변수 초기화

주목

생성자가 10번 실행되어
numOfCircles = 10 이 됨

numOfCircles = 0 이 됨

numOfCircles = 1 이 됨

numOfCircles = 2 가 됨

static 멤버 함수는 static 멤버만 접근 가능

40

- ▣ static 멤버 함수가 접근할 수 있는 것
 - static 멤버 함수
 - static 멤버 변수
 - 함수 내의 지역 변수
- ▣ static 멤버 함수는 non-static 멤버에 접근 불가 X
 - 객체가 생성되지 않은 시점에서 static 멤버 함수가 호출될 수 있기 때문

static 멤버 함수 getMoney()가 non-static 멤버 변수 money를 접근하는 오류

41

```
class PersonError {  
    int money;  
public:  
    static int getMoney() { return money; }  
    void setMoney(int money) { // 정상 코드  
        this->money = money;  
    }  
};  
  
int main(){  
    int n = PersonError::getMoney();  
  
    PersonError errorKim;  
    errorKim.setMoney(100);  
}
```

컴파일 오류.
static 멤버 함수는
non-static 멤버에 접근
할 수 없음.

non-static 멤버 변수

main()이 시작하기 전

```
static int getMoney() {  
    return money;  
}
```

money는 아직 생
성되지 않았음.

n = PersonError::getMoney();

```
static int getMoney() {  
    return money;  
}
```

생성되지 않는 변수를 접
근하게 되는 오류를 범함

PersonError errorKim;

errorKim

```
static int getMoney() {  
    return money;  
}
```

money

setMoney() { ... }

errorKim 객체가 생길 때
money가 비로소 생성됨

non-static 멤버 함수는 static에 접근 가능

42

```
class Person {  
    public: double money; // 개인 소유의 돈  
    static int sharedMoney; // 공금  
    ....  
    int total() { // non-static 함수는 non-static이나 static 멤버에 모두 접근 가능  
        return money + sharedMoney;  
    }  
};
```

non-static static

static 멤버 함수는 this 사용 불가

43

- static 멤버 함수는 객체가 생기기 전부터 호출 가능
 - ▣ static 멤버 함수에서 this 사용 불가

```
class Person {  
public:  
    double money; // 개인 소유의 돈  
    static int sharedMoney; // 공금  
    ....  
    static void addShared(int n) { // static 함수에서 this 사용 불가  
        this->sharedMoney += n; // this를 사용하므로 컴파일 오류  
    }  
};
```

sharedMoney += n;으로 하면 정상 컴파일