05) 함수와 참조, 복사 생성자

5.1) 함수의 인자 전달 방식 리뷰

- **값에 의한 호출, call by value**
  - 호출하는 코드에서 넘어온 **값이** 매개 변수에 복사됨
  - **ex)**

```
void swap(int a, int b){
    ...
}
int main(){
    ...
    swap(m, n); // 값을 복사하여 전달
}
```

- **주소에 의한 호출, call by address**
  - 호출하는 코드에서 넘어온 **주소 값이** 매개 변수에 저장됨
  - **ex)**

```
void swap(int *a, int *b){
    ...
}
int main(){
    ...
    swap(&m, &n);   // 값의 주소를 전달
}
```

CHECK TIME

1. 다음 빈칸을 채워 넣어라.

   '값에 의한 호출'은 실인자의 **값**을 함수 매개 변수에 **복사**하므로 매개 변수와 실인자는 서로 공간을 **공유하지 않는**다. '주소에 의한 호출'은 함수 호출 시 **주소**가 매개 변수로 전달되므로, 함수 내에서 **포인터 타입**에 매개 변수를 이용하여 실인자의 값을 **변경**할 수 있다.

5.1) 함수 호출시 객체 전달

'값에 의한 호출'로 객체 전달

- **ex)**

```
int main(){
    Circle waffle(30);
    increase(waffle);   // 객체를 복사하여 전달
    ...
}
void increase(Circle c){
    int r = c.getRadius();
    c.setRadius(r+1);
}
```

- **'값에 의한 호출' 로 객체를 전달할 때 문제점**
  - 매개 변수 객체의 **생성자는 실행되지 않고 소멸자만 실행**
  - **ex)**

```
...
int main(){
    Circle waffle(30);
    increase(waffle);
    cout << waffle.getRadius() << endl;
}
```

실행결과

```
생성자 실행 radius = 30      // waffle 생성
소멸자 실행 radius = 31      // c 소멸, c의 생성자 실행 X
30
소멸자 실행 radius           // waffle 소멸
```

'주소에 의한 호출' 로 객체 전달

- **ex)**

```
int main(){
    Circle waffle(30);
    increase(&waffle);  // 주소 전달
    cout << waffle.getRadius();
}
void increase(Circle *p){
    int r = p->getRadius();
    p->setRadius(r+1);
}
```

- **'주소에 의한 호출' 의 특징**
  - 생성자 소멸자의 비대칭 문제가 없다.

CHECK TIME

1. 클래스 Sample 타입의 매개 변수를 가지는 다음 함수 f( ) 에 대한 설명으로 틀린 것은?

```
void f(Sample a);
```

    1. 함수 f( ) 를 호출하면 객체 a가 생성된다.
    2. **함수 f( ) 를 호출하면 객체 a의 생성자가 실행된다. (X , 생성자 X)**
    3. 함수 f( ) 를 종료하면 객체 a의 소멸자가 실행된다.
    4. 함수 f( ) 를 호출하면 '값에 의한 호출' 이 일어난다.

2. 함수에 객체를 전달하는 경우, '갑에 의한 호출' 과 '주소에 의한 호출' 중 호출에 따른 비용 부담이 작은 것은?
주소에 의한 호출

5.3) 객체 치환 및 객체 리턴

객체 치환

- **객체끼리 치환**

```
Circle c1(5);
Circle c2(30);
c1 = c2;    // c2 객체를 c1 객체에 복사한다.
```

함수의 객체 리턴

- **객체를 리턴**

```
Circle getCircle(){
    Circle tmp(30);
    return tmp;     // 객체 tmp 리턴
}
int main(){
    Circle c;   // c의 반지름은 1
    c = getCircle();    // tmp 객체의 복사본이 c에 치환된다. c의 반지름이 30이 된다.
}
```

5.4) 참조와 함수

참조 변수

- **참조 변수 선언**
    - 참조자 **&**의 도입
    - 선언시 반드시 **원본 변수로 초기화** 필요
    - 이미 존재하는 변수에 대한 **다른 이름(별명)을 선언**

- 새로운 공간을 할당하지 않는다.
- 기존 변수를 공유한다.

- **ex)**

```
int n = 2;
int &refn = n;  // 참조 변수 refn 선언. refn은 n에 대한 별명. refn 과 n은 동일한 변수

Circle circle;
Circle &refc = circle;  // 참조 변수 refc 선언. refc는 circle에 대한 별명. refc와 circle은 동일한 변수
```

- **예제 5-3) 기본 타입 변수에 대한 참조**

```
#include <stdlib.h>
#include <iostream>
using namespace std;

int main() {
    cout << "i" << '\t' << "n" << "\t" << "refn" << endl;
    int i = 1;
    int n = 2;
    int &refn = n;
    n = 4;
    refn++;
    cout << i << "\t" << n << "\t" << refn << endl;

    refn = i;
    refn++;
    cout << i << "\t" << n << "\t" << refn << endl;

    int *p = &refn;
    *p = 20;
    cout << i << "\t" << n << "\t" << refn << endl;
    system("pause");
}
```

실행 결과

```
i       n       refn
1       5       5
1       2       2
1       20      20
```

CHECK TIME

1. public 속성의 show( ) 멤버 함수를 가지고 있는 클래스 Sample의 객체 a가 선언되어 있다.

   (1) 다음 중 틀린 선언문은?

   1. **Sample &p = *a; (X , 불가능 하다)**

2. Sample *q = &a;
3. Sample r = a;
4. Sample &s = a;

(2) 다음 주석에 따라 간단한 코드를 작성하라.

```
Sample &x = a    // 객체 a에 대한 참조 변수 x 선언
Sample *y = &a   // 객체 a에 대한 포인터 변수 y 선언
x.show() // 변수 x를 이용하여 show() 함수 호출
y->show()    // 변수 y를 이용하여 show() 함수 호출
```

참조에 의한 호출, call by reference

- **참조 매개 변수, 실인자와 공간을 공유**
- **ex)**

```
void swap(int &a, int &b){  // 참조 매개 변수 a, b
    ...
}
int main(){
    int m = 2, n = 9;
    swap(m,n);  // m, n 에 대한 참조 변수 a, b가 생긴다.
}
```

참조에 의한 호출로 객체 전달

- **Call by value 유의 사항**
  - 원본 객체를 변경시키지 않는다.
  - 생성자와 소멸자가 비대칭 구조로 작동
- **Call by reference 특징 정리**
  - **원본 객체에 대한 연산이 된다.**
  - **생성자 소멸자가 실행되지 않는다.**

참조 리턴

- **C++ 의 함수 리턴**
  - 함수는 값 외에 **참조 리턴 가능**
  - **참조 리턴** : 변수 등과 같이 현존하는 **공간에 대한 참조** 리턴 (변수의 값 리턴 X)
- **ex)**

```
char c = 'a';
char& find(){   // char 타입의 참조 리턴
    return c;   // 변수 c에 대한 참조 리턴
}
char a = find();        // a = 'a' 가 됨
char &ref = find();     // ref는 c에 대한 참조
ref = 'M';      // c = 'M'
find() = 'b';   // c = 'b'가 됨
```

- **예제 5-8) 참조 리턴**

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

char& find(char s[], int index) {
    return s[index];    // s[index] 공간의 참조 리턴
}

int main() {
    char name[] = "Mike";
    cout << name << endl;

    find(name, 0) = 'S';    // find()가 리턴한 위치에 문자 'S' 저장
    cout << name << endl;

    char& ref = find(name, 2);  // ref 는 name[2] 참조
    ref = 't';
    cout << name << endl;
}
```

실행 결과

```
Mike
Sike
Site
```

CHECK TIME

1. 다음 함수에 대한 호출이 잘된 것은?

```
int m = 3, n = 2;
void f(int &a, int b);
```

1. **f(m, 2); (O)**
2. f(&m, n);
3. f(*m, 2);
4. f(2, m);

2. 다음 함수들에 대해 문제에서 주어진 함수의 실행 결과 n 값은 무엇인가?

```
void f(int a){ a= -a; }
void g(int *a){ *a = -*a; }
void h(int &a){ a = -a; }
int n = 5;
```

1. f(n); **5 (값 복사)**
2. g(&n); **-5 (주소 전달)**
3. h(n); **-5 (참조)**

3. 다음 코드에 대해 아래 문제가 순서대로 실행될 때, 배열 ar은 어떻게 변하는가?

```
int ar[] = {0, 1, 3, 5, 7};
int& f(int n){
    return ar[n];
}
```

1. f(0) = 100; **{100, 1, 3, 5, 7}**
2. f(0) = f(1) + f(2) + f(3) + f(4); **{16, 1, 3, 5, 7}**
3. int& v = f(2); v++; **{16, 1, 4, 5, 7}**

5.5) 복사 생성자

복사 생성 및 복사 생성자

- **복사 생성** : 객체가 생성될 때 원본 객체를 복사하여 생성
- **복사 생성자** : 객체의 **복사 생성시** 호출되는 특별한 생성자 **(깊은 복사)**
- **특징**
  - 한 클래스에 **오직 한 개**만 선언 가능 (다중 선언이 안됨)
  - 자기 클래스에 대한 참조로 선언
  - 클래스에 대한 **참조 매개 변수**를 가지는 독특한 생성자
- **복사 생성자 선언** 예시)

```
class Circle{
    ...
    Circle(Circle& c);  // 복사 생성자 선언
    ...
};
Circle::Circle(Circle& c){      // 복사 생성자 구현
    this->radius = c.radius;
}

int main(){
    Circle src(30);     // 보통 생성자 호출
    Circle dest(src);   // src 객체를 복사하여 dest 객체 생성.
}
```

디폴트 복사 생성자

- **ex) 디폴트 복사 생성자**

```
Circle::Circle(Circle& s){      // 디폴트 복사 생성자
    this->radius = c.radius;    // 원본 객체 c의 각 멤버를 사본(this)에 복사한다.
}
```

> 디폴트 복사 생성자는 **얕은 복사**를 실행하도록 한다.

- **문제점**
  - 메모리를 반환할 때, 이미 반환한 메모리를 다시 반환하게 되므로, 오류가 발생하고 프로그램이 **비정상 종료**된다.

사용자 복사 생성자 작성

- **예제 5-11) 깊은 복사 생성자를 가진 정상적인 Person 클래스**

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

class Person {      // Person 클래스 선언
    char *name;
    int id;
public:
    Person(int id, char* name);     // 생성자
    Person(Person& person);         // 복사 생성자
    ~Person();      // 소멸자
    void changeName(char *name);
    void show() {
```

```
        cout << id << ',' << name << endl;
    }
};

Person::Person(int id, char* name) {    // 생성자
    this->id = id;
    int len = strlen(name);          // name 문자 개수
    this->name = new char[len + 1]; // name 문자열 공간 할당
    strcpy(this->name, name);        // name에 문자열 복사
}

// 깊은 복사 생생자
Person::Person(Person& person) {    // 복사 생성자
    this->id = person.id;   // id 값 복사
    int len = strlen(person.name);  // name의 문자 개수
    this->name = new char[len + 1]; // name을 위한 공간 할당
    strcpy(this->name, person.name);    // name의 문자열 복사
    cout << "복사 생성자 실행. 원본 객체의 이름" << this->name << endl;
}

Person::~Person() {     // 소멸자
    if (name)   // 만일 name에 동적 할당된 배열이 있으면
        delete[] name;  // 동적 할당 메모리 소멸
}

void Person::changeName(char* name) {   // 이름 변경
    if (strlen(name) > strlen(this->name))
        return; // 현재 name에 할당된 메모리보다 긴 이름으로 바꿀 수 없다.
    strcpy(this->name, name);
}

int main() {
    Person father(1,"Kitae");   // (1) father 객체 생성
    Person daughter(father);    // (2) daughter 객체 복사 생성. 복사 생성자 호출

    cout << "daughter 객체 생성 직후" << endl;
    father.show();      // (3) father 객체 출력
    daughter.show();    // (3) daughter 객체 출력

    daughter.changeName("Grace");           // (4) daughter의 이름을 "Grace"로 변경
    cout << "daughter 이름을 Grace로 변경";   // (5) father 객체 출력
    father.show();      // (5) father 객체 출력
    daughter.show();    // (5) daughter 객체 출력
}
```

실행 결과

```
복사 생성자 실행. 원본 객체의 이름 Kitae
daughter 객체 생성 직후
1. Kitae
1. Kitae
daughter 이름을 Grace로 변경
1. Kitae
1. Grace
```

묵시적 복사 생성

- **묵시적 복사 생성**
  1. 객체로 초기화하여 객체가 생성될 때

```
Person son = father;    // 복사 생성자 자동 호출
Person son;
son = father;   // 복사 생성자 호출되지 않음
```

  2. '값에 의한 호출' 로 객체가 전달될 때

```
void f(Person person){      // 매개 변수 person이 생성될 때 복사 생성자 호출
    ...
}
Person father(1, "Kitae");
f(father);  // '값에 의한 호출'로 father 객체 전달
```

  3. 함수가 객체를 리턴할 때

```
Person g(){
    Person mother(2, "Jane");
    return mother;  // mother의 복사본을 생성하여 복사본 리턴. 사본이 만들어질 때 복사
생성자 호출
}
g();
```

실습

참조 매개 변수를 가진 함수 만들기 연습

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

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

```
    }
    Circle(int radius) {
        this->radius = radius;
    }
    void setRadius(int radius) {
        this->radius = radius;
    }
    double getArea() {
        return 3.14 * radius * radius;
    }
};

void readRadius(Circle* c) {    // 주소 값을 받는다
    int r;
    cout << "정수 값으로 반지름을 입력하세요 >> ";
    cin >> r;           // 반지름 값 입력
    c->setRadius(r);    // 객체 c에 반지름 설정
}

int main() {
    Circle donut;
    readRadius(&donut);
    cout << "donut의 면적 = " << donut.getArea() << endl;
}
```

실행 결과

```
정수 값으로 반지름을 입력하세요 >> 3
donut의 면적 = 28.26
```

Book 클래스의 생성자, 소멸자, set( ) 함수, 복사 생성자 구현

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
#include <string>
using namespace std;

class Book {
    char* title;
    int price;
public:
    Book(char* title, int price);
    ~Book();
    Book(Book &c);      // 깊은 복사, 복사 생성자 작성
    void set(char *title, int price);
    void show() {
        cout << title << '.' << price << " 원" << endl;
```

```
    }
};

Book::Book(char* title, int price)  // 생산자
{
    this->price = price;
    int len = strlen(title);
    this->title = new char[len + 1];
    strcpy(this->title, title);
}

Book::~Book() {
    if (title)
        delete[] title;
}

Book::Book(Book &b) {
    int len = strlen(b.title);
    title = new char[len + 1];
    strcpy(title, b.title);
    price = b.price;
}

void Book::set(char *title, int price) {
    if (this->title)
        delete[] this->title;
    int len = strlen(title);
    this->title = new char[len + 1];
    strcpy(this->title, title);
    this->price = price;
}


int main() {
    char title[30] = "명품C++";
    Book cpp(title, 10000);
    Book java = cpp;
    strcpy(title, "명품자바");
    java.set(title, 12000);
    cpp.show();
    java.show();
}
```

실행 결과

```
명품C++.10000 원
명품자바.12000 원
```

Book 클래스의 생성자, 소멸자, set( ) 함수, 복사 생성자 구현(2)

```cpp
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <string>
using namespace std;

class Book {
    string title;   // string 선언
    int price;
public:
    Book(string title, int price);
    Book(Book &c);      // 깊은 복사, 복사 생성자 작성
    void set(string title, int price);
    void show() {
        cout << title << '.' << price << " 원" << endl;
    }
};

Book::Book(string title, int price) // 생산자
{
    this->price = price;
    this->title = title;
}

Book::Book(Book &b) {
    this->price = b.price;
    this->title = b.title;
}

void Book::set(string title, int price) {
    this->title = title;
    this->price = price;
}


int main() {
    Book cpp("명품C++", 10000);
    Book java = cpp;
    java.set("명품자바", 12000);
    cpp.show();
    java.show();
    system("pause");
}
```

실행결과

```
명품C++.10000 원
명품자바.12000 원
```

연습문제

이론문제

1. C++의 함수 인자 전달 방식이 아닌 것은?
    1. 값에 의한 호출
    2. 주소에 의한 호출
    3. 참조에 의한 호출
    4. **묵시에 의한 호출 (X)**

2. 일반적으로 함수 호출 시 가장 비용(cost) 부담이 큰 것은?
    1. **값에 의한 호출 (값을 복사하기 때문에)**
    2. 주소에 의한 호출
    3. 참조에 의한 호출
    4. 묵시에 의한 호출

3. 다음에서 f( ) 함수가 호출될 때 사용되는 인자 전달 방식은 무엇인가?

```
void f(int n[]);
int main(){
    int m[3] = {1, 2, 3};
    f(m);
}
```

주소에 의한 호출 (배열은 주소로 전달된다!!)

4. 다음 두 함수 선언은 같은 것인가?
    1. void f(int p[ ]); 와 void f(int *p); **두 함수의 선언은 같다. (배열과 포인터는 같은 Call by address이다.**
    2. void f(int *p); 와 void f(int &p); **두 함수의 선언은 다르다. (*p는 포인터 이고 &p는 참조 이다.)**

5. 다음 프로그램의 실행 결과는 무엇인가?
    1.

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

void square(int n){
    n = n * n;
}
int main(){
    int m = 5;
    square(m);
    cout << m;
}
```

    2. **실행 결과**

```
5   // 값에 의한 호출
```

1.

```
#include <iostream>
using namespace std;
void square(int &n){
    n = n * n;
}
int main(){
    int m = 5;
    square(m);
    cout << m;
}
```

2. **실행 결과**

```
25  // 참조에 의한 호출
```

6. 다음 프로그램의 실행 결과는 무엇인가?

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

void square(int n[], int size){
    for(int i = 0 ; i < size ; i++)
        n[i] = n[i] * n[i];
}
int main(){
    int m[3] = {1, 2, 3};
    square(m, 3);
    for(int i = 0 ; i < 3 ; i++)
        cout << m[i] << ' ';
}
```

실행 결과

```
1 4 9
```

7. char 형 변수 c가 선언되어 있을 때, 참조 변수 r의 선언 중 틀린 것은?

1. char &r = c;
2. **char r& = c; (X , 틀린 선언)**
3. char& r = c;
4. char &r = c;

8. 변수 c에 'a' 문자를 기록하지 못하는 것은?

```
char c;
char *p = &c;
char q = c;
char &r = c;
```

1. c = ' a ';
2. **q = ' a '; (X , 이것은 q에 'a' 를 저장하는 것이다.)**
3. r = ' a ';
4. *p = ' a ';

9. 다음 중 컴파일 오류가 발생하는 문장은?

```
int n = 10;
int &refn;          // 참조할 변수를 적지 않았으므로 오류!!
refn = n;
refn++;
int & m = refn;
```

10. 다음의 각 문제가 별도로 실행될 때 array 배열은 어떻게 되는가?

```
int array[] = {0, 2, 4, 6, 8, 10, 12, 14, 16, 18};
int &f(int n){
    return array[n];
}
```

1. f(9) = 100; **{0, 2, 4, 6, 8, 10, 12, 14, 16, 100}**
2. for(int i = 1; i < 9 ; i++) f(i) = f(i) + 2; **{0, 4, 6, 8, 10, 12, 14, 16, 18, 18}**
3. int v = f(0); v = 100; **{0, 2, 4, 6, 8, 10, 12, 14, 16, 18}, int v 는 값만 참조된 것이므로 0이 되고 그 후 100으로 저장하였으므로 배열은 변하지 않고 v의 값만 0에서 100으로 변한다.**
4. f(f(2)) = 0; **{0, 2, 4, 6, 0, 10, 12, 14, 16, 18}**

11. 다음 copy( ) 함수는 src 값을 dest에 복사하는 함수이다.

```
void copy(int dest, int src){
    dest = src;
}
```

copy( ) 를 이용하여 b 값을 a에 복사하고자 하지만, b 값이 a에 복사되지 않는다.

```
int a = 4, b = 5;
copy(a, b); // b 값을 a에 복사
```

복사되지 않는 이유가 무엇인지 설명하고, 복사가 잘 되도록 copy( ) 함수만 고쳐라.

복사되지 않는 이유는 함수에 값을 복사해온 뒤 식을 처리하고 소멸하기 때문에 복사되지 않는다. (값에 의한 호출로 복사되지 않는다.)

정답 코드

```
void copy(int &dest, int src){  // 값이 바뀌는 매개변수만 참조한다.
    dest = src;
}
```

12. 비슷하게 생긴 다음 두 함수가 있다.

```
int& big1(int a, int b){
    if(a > b) return a;
    else return b;
}
int& big2(int& a, int& b){
    if(a > b) return a;
    else return b;
}
```

다음 코드를 실행하였을 때, x, y 의 값이 어떻게 변하는지 예측하고, 그 이유를 설명하라.

```
int x = 1, y = 2;
int& z = big1(x, y);
z = 100;
int& w = big2(x, y);
w = 100;
```

big1 이 실행되면 x = 1, y = 2 , z = 100이고 big2 가 실행되면 x = 1, y = 100, z = 100 으로 변한다. 왜냐하면 big1은 값에 의한 호출이므로 a와 b가 소멸하므로 x와 y가 변하지 않고 big2는 참조에 의한 호출이기 때문에 y가 변하게 된다.

13. MyClass 클래스의 기본 생성자(디폴트 생성자) 와 복사 생성자의 원형은 무엇인가?

기본 생성자

```
MyClass(){
    ...
}
```

복사 생성자

```
MyClass(MyClass &a){
    ...
}
```

14. 클래스 MyClass가 있다고 할 때, 복사 생성자가 필요한 경우가 아닌 것은?

    1. MyClass a = f( ); // f( ) 가 MyClass 객체를 리턴하는 경우
    2. **void f(MyClass *p); (X , p가 가리키는 Class를 불러와 처리만 하는 것 이기 때문에)**
    3. MyClass b = a; // a는 MyClass 타입

4. MyClass b(a) // a는 MyClass 타입

15. 다음 클래스에 대해 물음에 답하여라.

```
class MyClass{
    int size;
    int *element;
public:
    MyClass(int size){
        this->size = size;
        element = new int[size];
        for(int i = 0; i < size; i++) element[i] = 0;
    }
};
```

1. 적절한 소멸자를 작성하라.

```
MyClass::~MyClass(){
    if(element[])
        delete[] element;
}
```

2. 컴파일러가 삽입하는 디폴트 복사 생성자 코드는 무엇인가?

```
MyClass::MyClass(Myclass &c){
    size = c.size;
    element = c.element;
}
```

17. 다음 클래스에서 컴파일러가 삽입하는 디폴트 복사 생성자는 무엇인가?

```
class Student{
    string name;
    string id;
    double grade;
};
```

답

```
Student::Student(Student &s){
    this->name = s.name;
    this->id = s.id;
    this->grade = s.grade;
}
```

18. 다음 클래스에서 컴파일러가 삽입하는 디폴트 복사 생성자는 무엇인가?

```
class Student{
    string *pName;
    string *pId;
    double grade;
}
```

답

```
Student::Student(Student &s){
    this->pName = s.pName;
    this->pId = s.pId;
    this->grade = s.grade;
}
```

19. 문제 15의 클래스에 대해 다음 치환문이 있다면 어떤 문제가 발생하는가?

```
int main(){
    MyClass a(5), b(5);
    a = b;  // 여기
}
```

a.element 포인터는 b.element에 할당된 메모리를 가리키게 되어 a, b 객체는 동적 메모리를 공유하게 된다. 그래서 main( ) 이 종료 될 때 객체 b가 소멸되고 b를 가리키고 있는 a가 소멸될 때 에러가 발생한다.

06) 함수 중복과 static 멤버

6.1) 함수 중복

중복 함수 조건

- **이름이 동일**하여야 한다.
- 매개 변수 **타입**이나 매개 변수 **개수**가 달라야 한다.
- **리턴 타입**은 고려되지 않는다.

- **함수 중복의 성공 사례**

```
int sum(int a, int b, int c){   // 매개 변수 3개
    return a + b + c;
}
double sum(double a, double b){ // 매개 변수 2개에 double 형
    return a + b;
}
int sum(int a, int b){  // 매개 변수 2개에 int 형
    return a + b;
}
int main(){
    cout << sum(2, 5, 33);
```

```
    cout << sum(12.5, 33.6);
    cout << sum(2, 6);
}
```

- **함수 중복 실패 사례**

```
int sum(int a, int b){
    return a + b;
}
double sum(int a, int b){
    return (double)(a + b);
}
int main(){
    cout << sum(2, 5)
}
```

> 함수 둘다 매개 변수가 **int 형 2개** 이므로 실패!!

함수 중복의 편리함

- **이름을 구분지어 기억할 필요X**

생성자 함수 중복

- **ex)**

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

소멸자 함수 중복

- 근본적으로 **중복**은 **불가능**하다.

CHECK TIME

1. 함수 중복이 가능하지 않는 경우는?
    1. 클래스 바깥에 선언된 전역 함수들
    2. 생성자를 포함하여 클래스의 멤버 함수들
    3. 기본 클래스와 이를 상속받는 파생 클래스의 함수들
    4. **소멸자 (X , 소멸자는 중복 불가)**

2. 함수 중복의 성공 조건에 맞도록 빈칸을 채워라.

   함수들은 **함수명**이 같아야 한다. 함수들은 **매개 변수**의 개수나 타입이 달라야 한다.

6.2) 디폴트 매개 변수

디폴트 매개 변수 선언

- 디폴트 매개 변수는 **'매개 변수 = 디폴트 값'** 형태

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

- **ex)**

```
star(); // 매개 변수 a에 디폴트 값 5 자동 전달. star(5) 와 동일
star(10);   // 매개 변수 a에 10 전달
```

- **디폴트 매개 변수 사례**

```
void msg(int id, string text = "Hello");
msg(10);    // 오류 X
msg();      // 오류 O, id 값을 지정하지 않았기 때문
msg("Hello");   // 오류 O, id 값을 지정 X
```

- **디폴트 매개 변수에 관한 제약 조건**

  디폴트 매개 변수는 **끝 쪽에 몰려 선언** 되어야 한다.

  - **ex)**

```
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

앞에서부터 순서대로 함수의 매개 변수에 전달, **나머지는 디폴트 값으로 전달**한다.

- **여러 개의 디폴트 매개 변수를 가진 함수**

```
void square(int width=1, int height=1);
square();
square(5);
square(3, 8);
```

예제 6-4) 디폴트 매개 변수를 가진 함수 만들기 연습

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

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

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();
    f('%');
    f('@', 5);

    system("pause");
}
```

실행 결과

```
%%%%%%%%%%
@@@@@@@@@@
@@@@@@@@@@
@@@@@@@@@@
@@@@@@@@@@
@@@@@@@@@@
```

함수 중복 간소화

- **디폴트 매개 변수의 장점**

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

> 중복 함수들과 디폴트 매개 변수를 가진 함수를 함께 사용 불가!!

```
class Circle{
    ...
public:
    Circle(int r){
        radius = r;
    }
    Circle(int r=1){    // 오류!! 중복된 함수 선언 불가
        radius = r;
    }
};
```

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

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

```
void fillLine(){
    for(int i = 0 ; i < 25 ; i++)
        cout << '*';
    cout << endl;
}
void fillLine(int n, char c){
    for(int i = 0 ; i < n ; i++)
        cout << c;
    cout << endl;
}
```

  답

```
void fillLine(int n = 25, char c = '*'){ // n개의 c문자를 한라인으로
    for(int i = 0 ; i < n ; i++){
        cout << c;
    } cout << endl;
}
```

CHECK TIME

1. 다음 중 디폴트 매개 변수를 가진 함수 선언이 잘못된 것은?

1. void f(int a, int b=0);
2. void f(int a, int b, double d=0.0);
3. void f(int a, int b=0, double d=0.0);
4. **vid f(int a=0, int b, double d=0.0); (X , 디폴트 매개 변수는 끝쪽에만 !!)**

2. 디폴트 매개 변수를 가진 다음 함수 f( )를 잘못 호출한 것은?

```
void f(string name, string addr=" ", int id=2000);
```

1. **f( ); (X , name의 값을 지정 하지 않았다.)**
2. f("Grace");
3. f("Helen", "Seoul");
4. f("Ashley", "Gainesville", 2011);

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

```
int sum(int a, int b){
    return a + b;
}
int sum(int a){
    return a + 10;
}
```

답

```
int sum(int a, int b = 10){
    return a + b;
}
```

6.3) 함수 중복의 모호성

형 변환으로 인한 모호성

- **ex) 형변환**

```
double square(double a);
square(3);  // int 타입의 매개 변수 전달(자동 형변환), 컴파일 오류 X
```

> **char -> int -> long -> float -> double (자동 형 변환)**

- **ex) 모호한 예**

```
float square(float a);
double square(double a);
square(3);  // 정수 3을 float으로 형변환할지 double로 형변환할지 컴파일 오류!!!!
```

참조 매개 변수로 인한 모호성

- **ex) 참조 매개 변수**

```
int add(int a, int b);
int add(int a, int &b);
int s=10, t=20;
add(s, t);  // 컴파일 오류!! 함수 호출의 모호성
```

디폴트 매개 변수로 인한 모호성

- **ex) 디폴트 매개 변수**

```
void msg(int id);
void msg(int id, string s= " ");
msg(6); // 컴파일 오류!! 함수 호출 모호
```

- **틀린 함수 중복**

```
void f(int a[]);
void f(int *a);     // 배열과 포인터 공존 불가!!!
```

CHECK TIME

1. 다음 함수 중복 중에서 모호한 함수 호출의 가능성이 있는 경우는?
    1. **void f(int a, int b)); void f(int a=3, int b=3) (모호하다, 매개 변수의 개수와 형이 모두 같기 때문)**;
    2. void f(int a); void f(int* a);
    3. void f(int *a); void f(int &a);
    4. void f(int a, int b); void f(int a=3);

6.4) static 멤버

static의 특성

- **static**
    - 생명주기 : 프로그램이 시작부터 종료까지
    - 사용범위 : 선언된 범위

- **클래스의 멤버 (static 멤버)**
  - 프로그램이 시작할 때 생성
  - **클래스 당 하나만 생성**
  - **모든 인스턴스(객체)들이 공유하는 멤버**
- **ex) static 멤버 선언**

```
class Person{
public:
    double money;               // non-static 멤버 선언
    void addMoney(int money){
        this->money += money;
    }

    static int sharedMoney;     // static 멤버 선언 (전역 변수)
    static void addShared(int n){
        sharedMoney += n;
    }
};

int Person::sharedMoney = 10;   // 프로그램의 전역 공간에 선언, static 변수 공간 할당 (필수)
```

static 멤버 사용 : 객체의 멤버로 접근하는 방법

static 멤버는 **객체 이름**이나 **객체 포인터**를 이용하여 보통 멤버와 동일하게 다루면 된다.

- **ex)**

```
Person lee;
lee.sharedMoney = 500;  // 객체 이름으로 접근

Person *p;
p = &lee;
p->addShared(200);  // 객체 포인터로 접근
```

static 멤버 사용 : 클래스명과 범위지정 연산자(::) 로 접근

- **사용 방법**

```
Person::sharedMoney = 200;  // 클래스명으로 접근
// han.sharedMoney = 200; 위와 동일한 표현
Person::addShared(200);     // 클래스명으로 접근
// lee.addShared(200)l  위와 동일한 표현
```

- 그러나 non-static 멤버는 클래스명으로 **접근 불가!!**

```
Person::money = 100;    // 컴파일 오류
lee.money = 100;
Person *p = &lee;
p->addMoney(200);   // 가능
```

- **코드 사례**

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

class Person {
public:
    double money;
    void addMoney(int money) {
        this->money += money;
    }

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

// static 변수 생성, 전역 공간에 생성 ( 필수 !! )
int Person::sharedMoney = 10;

int main() {
    Person::addShared(50);
    cout << Person::sharedMoney << endl;

    Person han;
    han.money = 100;
    han.sharedMoney = 200;
    Person::sharedMoney = 300;
    Person::addShared(100);

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

    system("pause");
}
```

실행결과

```
60
100 400
```

static의 활용

- 전역 변수나 전역 함수를 클래스에 캡슐화

```
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;
}   // 캡슐화X, 전역 함수들이 존재하는 좋지 않은 코드 사례
```

- static 멤버를 가진 Math 클래스 작성

```
class Math{
public: // 주목!!
    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;      // Math:: 주목!!
    cout << Math::max(10, 8) << endl;   // Math:: 주목!!
}
```

- 객체 사이에 공유 변수를 만들고자 할 때

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

class Circle {
private:
    static int numOfCircles;
    int radius;
public:
    Circle(int r = 1);
    ~Circle() {
        numOfCircles--;
```

```
    }
    double getArea() {
        return 3.14*radius*radius;
    }
    static int getNumOfCircles() {
        return numOfCircles;
    }
};

Circle::Circle(int r) {
    radius = r;
    numOfCircles++;
}

int Circle::numOfCircles = 0;   // 주목!!

int main() {
    Circle *p = new Circle[10];

    // Circle::getNumOfCircles() 주목!!

    cout << "생존하고 있는 원의 개수 = " << Circle::getNumOfCircles() << endl;

    delete[] p;
    cout << "생존하고 있는 원의 개수 = " << Circle::getNumOfCircles() << endl;

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

    system("pause");
}
```

실행결과

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

static 멤버 함수의 특징

- **static 멤버 함수는 오직 static 멤버들만 접근**

  static 멤버 함수에서 non-static 멤버에 접근하는 것은 허용되지 않는다.

```
class PersonError{
    int money;
public:
    static int getMoney(){
        return money;   // 컴파일 오류. static 멤버 함수는 non-static 멤버에 접근할 수 없다.
    }
    ...
}
```

그 반대로 non-static 맴버 함수는 static 멤버를 접근하는데 제약이 없다.

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

static 멤버 함수는 this를 사용할 수 없다.

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

CHECK TIME

1. static, 인스턴스 중에서 골라 빈칸에 기입하라.

   인스턴스 멤버 션수는 객체가 생성될 때 객체 내에 메모리 공간을 할당받지만, **static** 멤버 변수는 프로그램이 시장할 때 객체 외부에 메모리 공간을 할당받아 생긴다. 그러므로 **static** 멤버 변수는 동일한 클래스 타입의 모든 객체들에 의해 공유된다.

1. 다음 코드에서 틀린 부분을 찾고 이유를 설명하라.

```
class Sample{
    static int a;
    int b;
public:
    void f(){ a = 3;}
    void g(){ b = 3;}
    static void h(){ a = 3;}
    static void s(){ b = 3;}    // static은 non-static을 접근할 수 없다.
};
```

1. 다음 코드에서 틀린 부분을 찾고 이유를 설명하라.

```
class Sample{
public:
    static int a;
    int b;
    void f();
    static void h();
};

int Sample::a;

int main(){
    Sample sample;
    sample.b = 5;
    sample.a = 5;
    Sample::f();    // non-static은 접근 X
    Sample::h();
}
```