

I2PXO2CUBE

Week 4

방찬웅

C++ STUDY

Week 4
함수와 참조, 복사 생성자

01

하나,

02

둘,

03

셋,

Call by value
Call by reference

참조

복사 생성자

01

하나,

Call by value
Call by reference

02

둘,

참조

03

세엣,

복사 생성자

인자 전달 방식

call by value : 값에 의한 호출

call by reference : 주소에 의한 호출



실습 (1)

Swap() 구현

오랜만에 헬프

실습 (1)

Swap() 구현

오랜만에 헬프

Call by value (값에 의한 호출)

```
void increaseRadius(Circle c, float num)
{
    c.setRadius(c.getRadius() + num);
}

int main()
{
    Circle waffle(30);
    increaseRadius(waffle, 5.0f);
    cout << waffle.getRadius() << endl;
}
```

Call by value (값에 의한 호출) 결과

```
생성자 실행 radius : 30
소멸자 실행 radius : 35
30
소멸자 실행 radius : 30
```

문제점은? 그리고 이유는? 그리고 해결하기 위한 방법은?

Call by reference (주소에 의한 호출)

```
void increaseRadius(Circle* c, float num)
{
    c->setRadius(c->getRadius() + num);
}
```

```
int main()
{
    Circle waffle(30);
    increaseRadius(&waffle, 5.0f);
    cout << waffle.getRadius() << endl;
}
```

Call by reference (주소에 의한 호출) 결과

```
생성자 실행 radius : 30  
35  
소멸자 실행 radius : 35
```

결과는? 주의해야 할 점은?

객체 치환

```
// 객체 치환
Circle c1;
Circle c2(10);
cout << c1.getRadius() << endl;
cout << c2.getRadius() << endl;
c1 = c2;
cout << c1.getRadius() << endl;
cout << c2.getRadius() << endl << endl;
```

모든 데이터가 비트 단위로 복사된다.

내용은 완전히 같지만 c1, c2는 별개의 객체이다.

```
생성자 실행 radius : 1
생성자 실행 radius : 10
1
10
10
10

소멸자 실행 radius : 10
소멸자 실행 radius : 10
```

객체 치환

```
// 객체 치환
Circle c1;
Circle c2(10);
cout << c1.getRadius() << endl;
cout << c2.getRadius() << endl;
c1 = c2;
cout << c1.getRadius() << endl;
cout << c2.getRadius() << endl << endl;
```

모든 데이터가 비트 단위로 복사된다.

내용은 완전히 같지만 c1, c2는 별개의 객체이다.

```
생성자 실행 radius : 1
생성자 실행 radius : 10
```

1

10

10

10

```
소멸자 실행 radius : 10
```

```
소멸자 실행 radius : 10
```

객체 리턴

```
Circle getPizza()
{
    Circle pizza(50);
    return pizza;
}
```

```
// 객체 리턴
Circle c;
cout << c.getRadius() << endl;
c = getPizza();
cout << c.getRadius() << endl << endl;
```

01

하나,

Call by value
Call by reference

02

둘,

참조

03

셋엣,

복사 생성자

참조(reference) &



유재석 == 메뚜기

메뚜기는 유재석의 별명

참조란 가리킨다는 뜻, 참조 변수는 선언된 변수에 대한 별명

참조(reference)

```
int main()
{
    int n = 2;
    int& refn = n; // 참조 변수 refn 선언.
    // refn은 n에 대한 별명, 즉 동일한 변수이다.
    cout << "n = " << n << endl;
    refn = 5;
    cout << "n = " << n << endl;
    // 참조 변수에 대한 포인터 생성 가능
    int* p = &refn;
    *p = 7;
    cout << "n = " << n << endl;
```

포인터 생성도 가능하다

선언 시 반드시 원본 변수로 초기화

원본 변수의 공간을 공유한다

참조 변수의 참조 변수 생성도 가능하다

참조(reference)

```
Circle circle;
Circle& refc = circle; // 참조 변수 refc 선언.
// refc은 circle에 대한 별명, 즉 동일한 변수이다.
cout << "circle의 반지름 : " << circle.getRadius() << endl;
refc.setRadius(10);
cout << "circle의 반지름 : " << circle.getRadius() << endl;
```

객체 참조 변수도 동일

참조에 의한 호출 (call by reference)

C++의 새로운 인자 전달 방식

매개 변수를 참조 타입으로 선언하여, 매개 변수가 함수를 호출하는 쪽의 "실인자"를 참조하여 "실인자"와 공간을 공유하도록 하는 인자 전달 방식

참조에 의한 호출 (call by reference)

```
void swap(int& a, int& b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}
```

훨씬 간단하고 깔끔해진 함수

참조에 의한 호출의 장점

**주소에 의한 호출은 & 연산자와 같은 * 기호 사용으로
실수의 가능성이 커지고 가독성이 떨어진다.**

**참조에 의한 호출은 간단히 변수를 넘겨주기만 하면되고,
함수 내에서도 참조 매개변수를 보통 변수처럼 사용하기 때문에
작성이 쉽고 보기 좋은 코드가 된다.**

유의사항 정리

값에 의한 호출

- 1) 함수 내에서 매개 변수 객체를 변경하여도, 원본 객체를 변경시키지 않는다.
- 2) 매개 변수 객체의 생성자가 실행되지 않고 소멸자만 호출되는 비대칭 구조

참조에 의한 호출

- 1) 참조 매개 변수로 이루어진 모든 연산은 원본 객체에 대한 연산이 된다.
- 2) 참조 매개 변수는 이름만 생성되므로, 생성자와 소멸자는 아예 실행되지 않는다.

실습(2)

참조 매개 변수를 가진 함수 만들기

반지를 값을 입력받고 Circle 객체에 반지를 설정하는 `readRadius()` 작성

```
Circle cir;
cout << "cir의 반지를 : " << cir.getRadius() << endl;
readRadius(cir);
cout << "cir의 반지를 : " << cir.getRadius() << endl;
```

```
cir의 반지를 : 1
설정할 반지를 입력 : 10
cir의 반지를 : 10
```

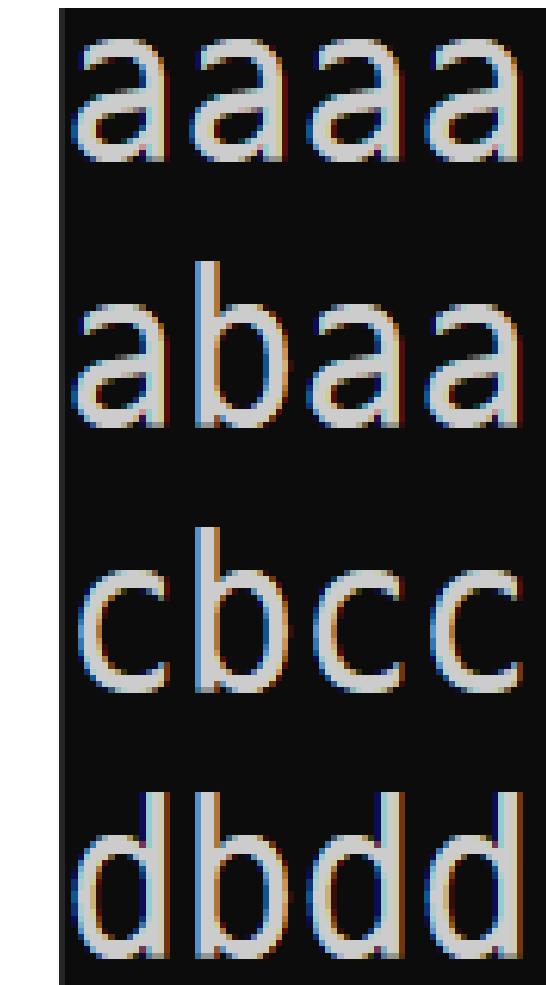
참조 리턴

C++에서는 함수가 참조를 리턴할 수 있다!

참조 리턴이란 변수 등과 같이 현존하는 공간에 대한 참조의 리턴이다.

```
char& find(char& c)
{
    return c;
}
```

```
// 참조 리턴
char c = 'a';
char ch = find(c);
char& ref = find(c);
cout << c << ch << ref << find(c) << endl;
ch = 'b';
cout << c << ch << ref << find(c) << endl;
ref = 'c';
cout << c << ch << ref << find(c) << endl;
find(c) = 'd';
cout << c << ch << ref << find(c) << endl;
```



aaaa
abaa
cbcc
dbdd

참조 리턴

```
char& find(char s[], int index)
{
    return s[index];
}
```

```
// 참조 리턴
char name[] = "Bang";
cout << name << endl;
find(name, 0) = 'P';
cout << name << endl;
char& refs = find(name, 1);
refs = 'o';
cout << name << endl;
```

Bang
Pang
Pong

따라서 참조 리턴이란 공간에 대한 참조를 반환 해주는 것!

01

하나,

Call by value
Call by reference

02

둘,

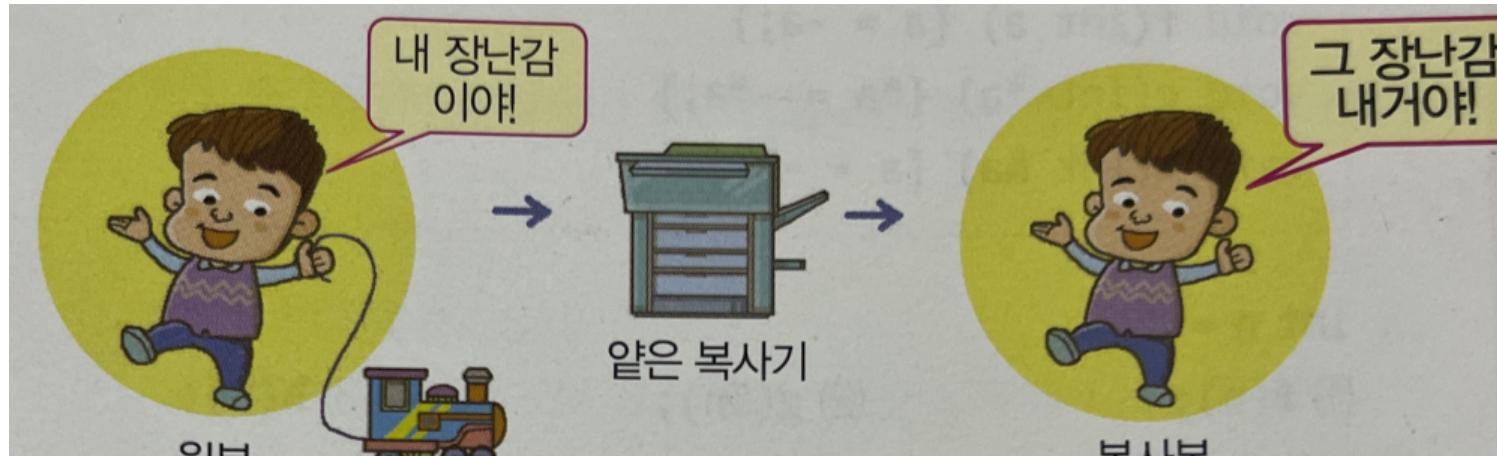
참조

03

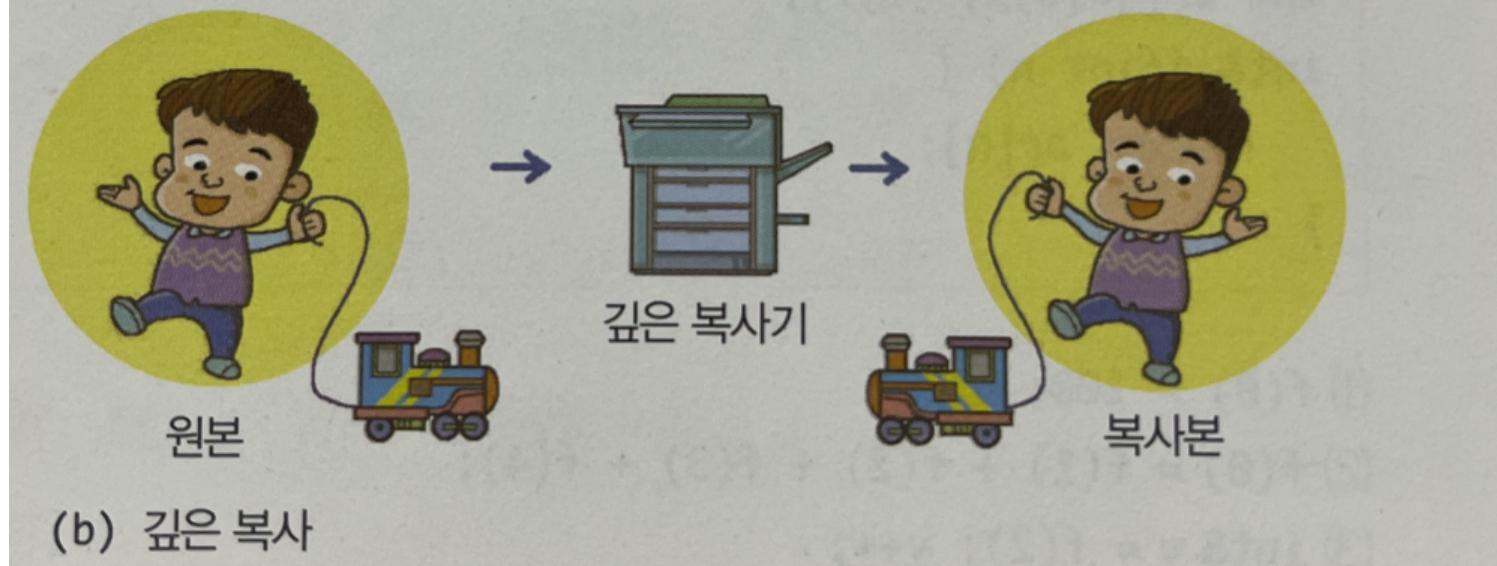
세엣,

복사 생성자

얕은 복사(shallow copy)와 깊은 복사(deep copy)



(a) 얕은 복사

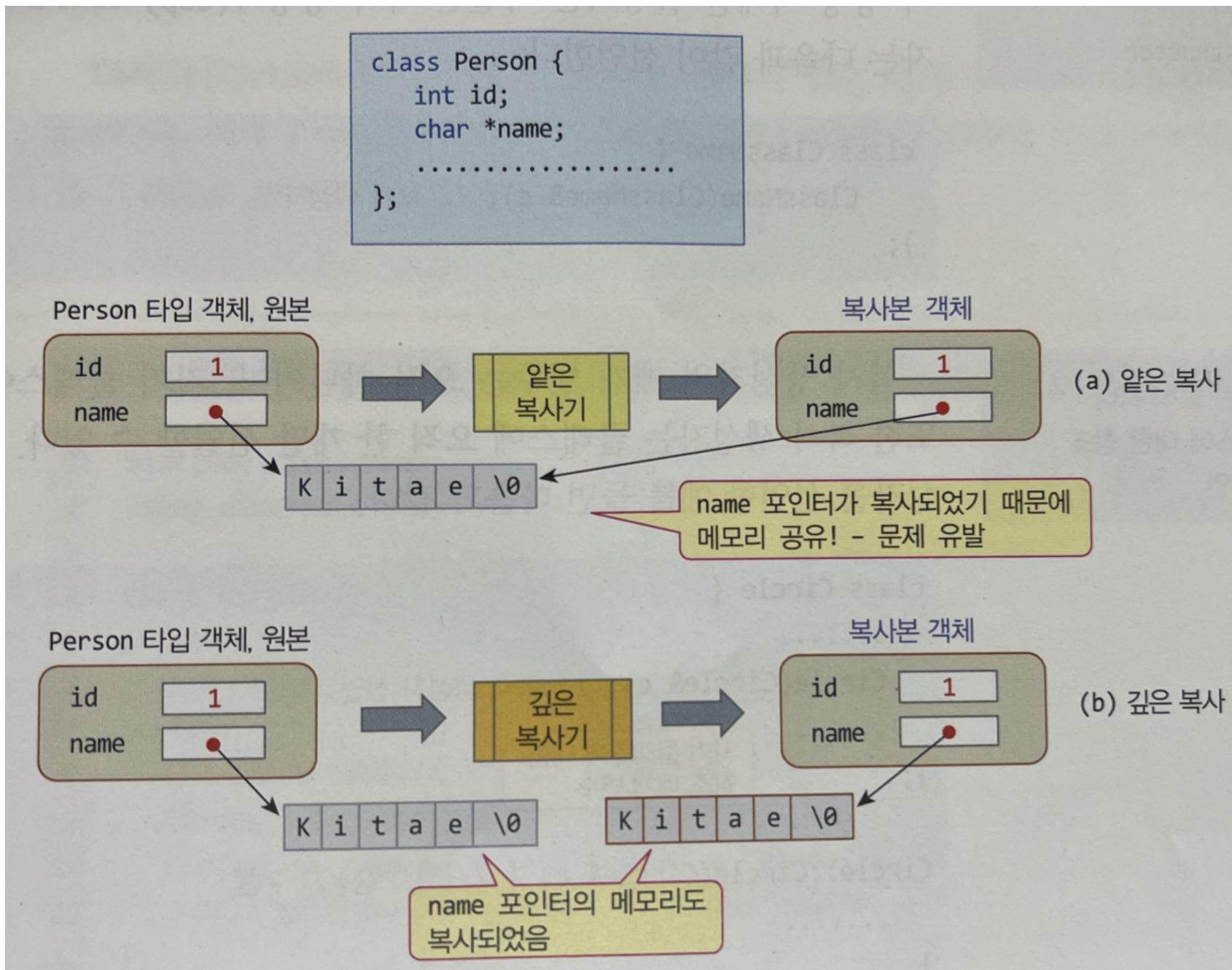


(b) 깊은 복사

서로 장난감을 자기 것으로 아는 충돌 발생

장난감도 복사되므로 충돌 X

얕은 복사와 깊은 복사



메모리를 공유한다.

별개의 메모리를 가리킨다.

얕은 복사와 깊은 복사

**얕은 복사가 일어나면 메모리를 공유하고 있기 때문에
사본에서 정보를 수정하면 원본 객체도 같이 수정이 된다.**

따라서, 가능한 얕은 복사가 일어나지 않도록 해야한다.

복사 생성 및 복사 생성자

복사 생성 : 객체가 생성될 때 원본 객체를 복사하여 생성되는 경우

C++에는 복사 생성 시에만 실행되는 특별한 "복사 생성자"가 있다.

```
class Circle
{
public:
    Circle();
    Circle(float radius);
    Circle(Circle& c); // 복사 생성자 정의
```

```
// 복사 생성자 구현
Circle::Circle(Circle& c)
{
    this->radius = c.radius;
    cout << "복사 생성자 실행 radius : " << this->radius << endl << endl;
}
```

```
int main()
{
    Circle c(30);
    Circle cir(c);
```

복사 생성 및 복사 생성자

복사 생성자를 가지고 있지 않은 클래스는 Default 복사 생성자가 호출된다.

원본 객체의 모든 멤버를 일대일로 사본에 복사하도록 구성 -> 얇은 복사

**포인터 타입의 멤버 변수가 없는 클래스의 경우, 얇은 복사는 문제가 없다.
일대일로 복사해도 공유의 문제가 발생하지 않기 때문이다.**

**포인터 멤버 변수를 가지고 있는 경우, 원본 객체의 포인터 멤버 변수가
사본 객체의 포인터 멤버 변수에 복사되면, 이 둘은 같은 메모리를 가리키게 되어
심각한 문제를 야기한다.**

포인터 멤버를 가지고 있는 클래스 예제(복사 생성자X)

```
class Person
{
public:
    Person(int id, char* name); // 기본 생성자
    //Person(Person& person); // 복사 생성자
    ~Person();
    void 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);
    }
    void show() { cout << id << ", " << name << endl; }

private:
    int id;
    char* name;
};
```

클래스

```
Person::Person(int id, char* name)
{
    this->id = id;
    int len = strlen(name);
    this->name = new char[len + 1];
    strcpy(this->name, name);
}
```

생성자

```
Person::~Person()
{
    if (name)
        delete[] name;
}
```

소멸자

포인터 멤버를 가지고 있는 클래스 예제(복사 생성자X)

```
Person::Person(int id, char* name)
{
    this->id = id;
    int len = strlen(name);
    this->name = new char[len + 1];
    strcpy(this->name, name);
}

Person::~Person()
{
    if (name)
        delete[] name;
}
```

```
int main()
{
    char name[] = "Bang";
    Person father(1, name);
    Person daughter(father);

    cout << "daughter 객체 생성 직후 -----" << endl;
    father.show();
    daughter.show();

    char name1[] = "Chan";
    daughter.changeName(name1);

    cout << "daughter 이름을 Chan으로 변경한 후 -----" << endl;
    father.show();
    daughter.show();
}
```

포인터 멤버를 가지고 있는 클래스 예제(복사 생성자O)

```
Person::Person(Person& person)
{
    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()
{
    if (name)
        delete[] name;
}
```

```
int main()
{
    char name[] = "Bang";
    Person father(1, name);
    Person daughter(father);

    cout << "daughter 객체 생성 직후 -----" << endl;
    father.show();
    daughter.show();

    char name1[] = "Chan";
    daughter.changeName(name1);

    cout << "daughter 이름을 Chan으로 변경한 후 -----" << endl;
    father.show();
    daughter.show();
}
```

묵시적 복사 생성

개발자도 모르게 복사 생성자가 호출되는 다른 경우들이 있다.

1) 객체를 생성할 때 객체로 초기화를 할 때

2) 값에 의한 호출로 객체가 전달될 때

3) 함수가 객체를 리턴할 때

실습(3)

누적해서 더하는 계산기 만들기 (누산기 : Accumulator)

```
class Accumulator
{
public:
    Accumulator(int value);
    Accumulator& add(int n);
    ~Accumulator();
    int get() { return value; }
private:
    int value;
};
```

```
int main()
{
    Accumulator acc(10);
    acc.add(5).add(10).add(20);
    cout << "총 합은 : " << acc.get() << endl;
}
```

I2PXO2CUBE

Week4

방찬웅

Q&A 및 토킹

감사합니다!