Chapter 05. const, friend, static

박 종 혁 교수

UCS Lab

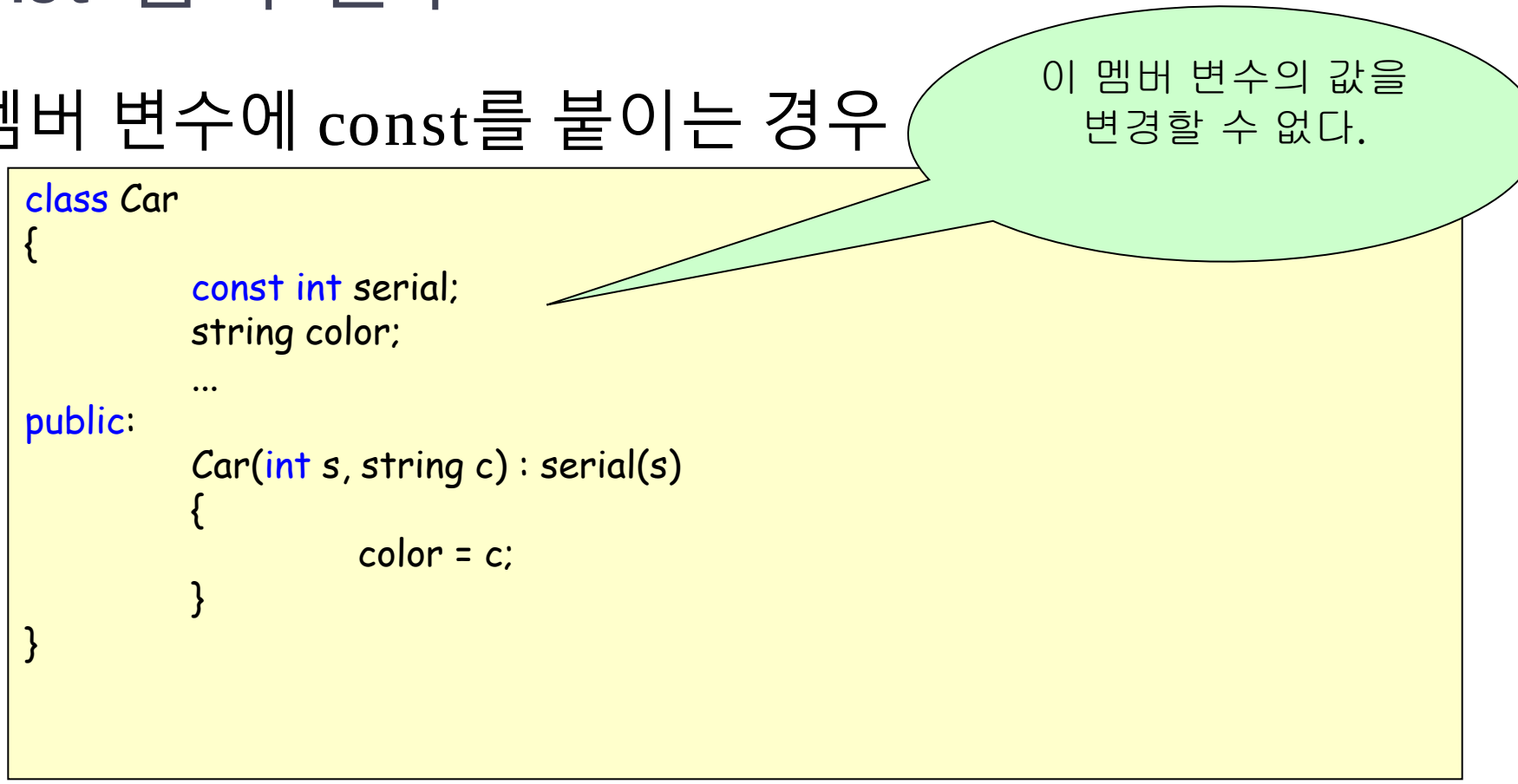
Tel: 970-6702

Email: jhpark1@seoultech.ac.kr

Chapter 05-1. const
멤버 변수, 함수, 객체

const 멤버 변수

- 멤버 변수에 const를 붙이는 경우




```
class Car
{
        const int serial;
        string color;
        ...
public:
        Car(int s, string c) : serial(s)
        {
                color = c;
        }
}
```

const 멤버 함수

- 멤버 함수의 상수화
  - 함수 내부에서 멤버변수의 값을 변경할 수 없음
  - 이 함수 내에서 상수화 되지 않은 함수의 호출을 허용하지 않음
  - 멤버변수의 포인터의 리턴을 허용하지 않음

- 사용 예

```
void ShowData()  const
{
        cout<<"이름: "<<name<<endl;
        cout<<"나이: "<<age<<endl;
        cout<<"학번: "<<id<<endl;
        cout<<"학과: "<<major<<endl;
}
```

- 멤버 함수에 const를 붙이는 경우

```
void displayInfo() const
{
                cout << "속도: " << speed << endl;
                cout << "기어: " << gear << endl;
                cout << "색상: " << color << endl;
}
```

이 함수 안에서는 멤버 변수의 값을 변경할 수 없다.

const 멤버 함수 예

```
#include <iostream>
using std::cout;
using std::endl;

class Count
{
	int cnt;
public :
	Count() : cnt(0){}
	int* GetPtr() const {
		return &cnt;	// Compile Error
	}

	void Increment(){
		cnt++;
	}

	void ShowData() const {
		ShowIntro();	// Compile Error
		cout<<cnt<<endl;
	}
	void ShowIntro() {
		cout<<"현재 count의 값 : "<<endl;
	}
};
```

```
int main()
{
	Count count;
	count.Increment();
	count.ShowData();

	return 0;
}
```

const 객체

- const 객체
  - 데이터의 변경이 허용되지 않는 객체
  - const 함수 이외에는 호출 불가
- const와 함수 오버로딩
  - const도 함수 오버로딩 조건에 포함

```
void function(int n) const { . . . . . }
void function(int n) { . . . . . }
```

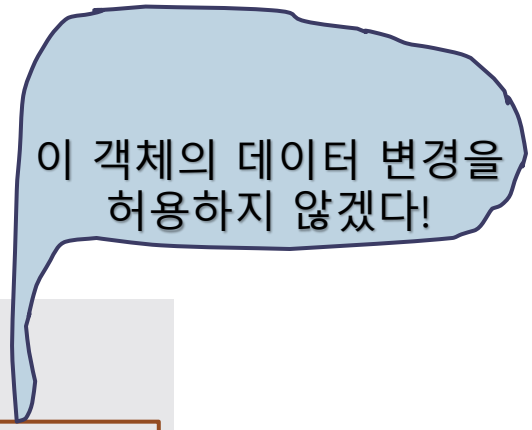
- 객체에 const를 붙이는 경우

```
int main()
{
        const Car c1(0, 1, "yellow");
        c1.setSpeed();      // 오류!
        return 0;
}
```

const로 선언된 객체를 대상으로는 const로 선언되지 않는 멤버함수의 호출이 불가능함

```
class SoSimple
{
private:
    int num;
public:
    SoSimple(int n) : num(n)
    { }
    SoSimple& AddNum(int n)
    {
        num+=n;
        return *this;
    }
    void ShowData() const
    {
        cout<<"num: "<<num<<endl;
    }
};
```

```
int main(void)
{
    const SoSimple obj(7);
    // obj.AddNum(20);
    obj.ShowData();
    return 0;
}
```



const와 함수 오버로딩

```
class SoSimple
{
private:
    int num;
public:
    SoSimple(int n) : num(n)
    { }
    SoSimple& AddNum(int n)
    {
        num+=n;
        return *this;
    }
    void SimpleFunc()
    {
        cout<<"SimpleFunc: "<<num<<endl;
    }
    void SimpleFunc() const
    {
        cout<<"const SimpleFunc: "<<num<<endl;
    }
};
```

함수의 const 선언 유무는 함수 오버로딩의 조건이 된다!

const 객체 또는 참조자를 대상으로 멤버함수 호출 시 const 선언된 멤버함수가 호출된다!

```
void YourFunc(const SoSimple &obj)
{
    obj.SimpleFunc();
}
int main(void)
{
    SoSimple obj1(2);
    const SoSimple obj2(7);
    obj1.SimpleFunc();
    obj2.SimpleFunc();
    YourFunc(obj1);
    YourFunc(obj2);
    return 0;
}
```

실행결과

```
SimpleFunc: 2
const SimpleFunc: 7
const SimpleFunc: 2
const SimpleFunc: 7
```

ConstOverloading.cpp

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

class SoSimple
{
private:
    int num;
public:
    SoSimple(int n) : num(n)
    { }
    SoSimple& AddNum(int n)
    {
            num+=n;
            return *this;
    }
    void SimpleFunc ()
    {
            cout<<"SimpleFunc: "<<num<<endl;
    }
    void SimpleFunc () const
    {
            cout<<"const SimpleFunc: "<<num<<endl;
    }
};

void YourFunc(const SoSimple &obj)
{
    obj.SimpleFunc();
}

int main(void)
{
    SoSimple obj1(2);
    const SoSimple obj2(7);

    obj1.SimpleFunc();
    obj2.SimpleFunc();

    YourFunc(obj1);
    YourFunc(obj2);
    return 0;
}
```

생각해봅시다

- 어느예제가 에러날까?
- 왜 에러가 날까?

예1)

```
#include <iostream>
using std::cout;
using std::endl;

class AAA
{
	int num;
public :
	AAA(int _num) : num(_num) {}
	void Add(int n){
		num+=n;
	}
	void ShowData(){
		cout<<num<<endl;
	}
};

int main()
{
	const AAA aaa(10);
	aaa.Add(10);

	return 0;
}
```

예2)

```
#include <iostream>
using std::cout;
using std::endl;
class AAA
{
	int num;
public :
	AAA(int _num) : num(_num) {}

	void ShowData(){
		cout<<"void ShowData() 호출"<<endl;
		cout<<num<<endl;
	}
	void ShowData() const {
		cout<<"void ShowData() const 호출"<<endl;
		cout<<num<<endl;
	}
};
int main()
{
	const AAA aaa1(20);
	AAA aaa2(70);
	 aaa1.ShowData();
	aaa2.ShowData();
	return 0;
}
```

생각해봅시다

- 어느예제가 에러날까?
- 왜 에러가 날까?

예1)

```
#include <iostream>
using std::cout;
using std::endl;

class AAA
{
	int num;
public :
	AAA(int _num) : num(_num) {}
	void Add(int n){
		num+=n;
	}
	void ShowData(){
		cout<<num<<endl;
	}
};

int main()
{
	const AAA aaa(10);
	aaa.Add(10);	// Compile Error
	aaa.ShowData();	// Compile Error

	return 0;
}
```

예2)

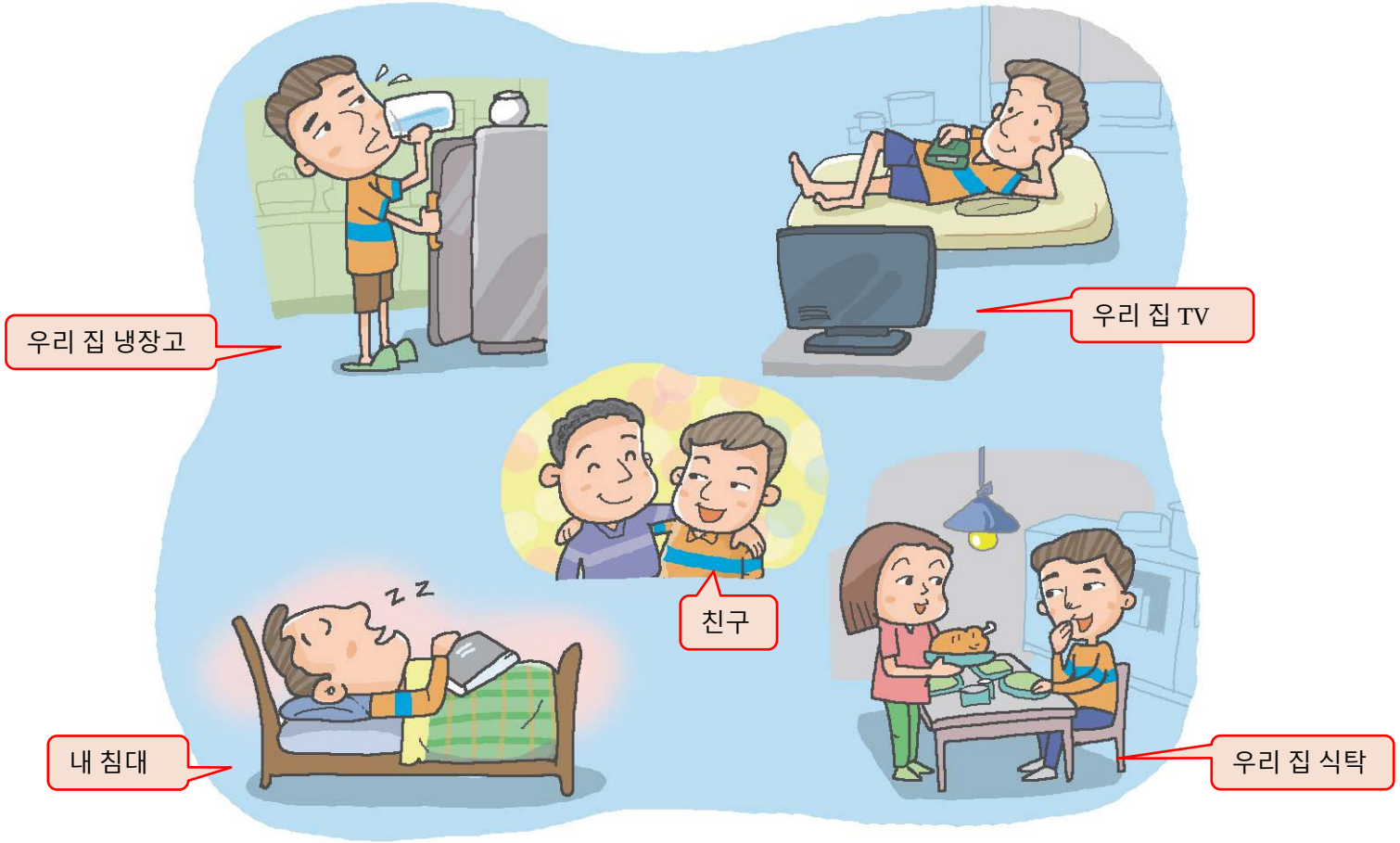
```
#include <iostream>
using std::cout;
using std::endl;
class AAA
{
	int num;
public :
	AAA(int _num) : num(_num) {}

	void ShowData(){
		cout<<"void ShowData() 호출"<<endl;
		cout<<num<<endl;
	}
	void ShowData() const {
		cout<<"void ShowData() const 호출"<<endl;
		cout<<num<<endl;
	}
};
int main()
{
	const AAA aaa1(20);
	AAA aaa2(70);
	 aaa1.ShowData();	// void ShowData() const  호출
	aaa2.ShowData();	// void ShowData() 호출
	return 0;
}
```

Chapter 05-2. friend 선언

친구란?

친구?

내 가족의 일원은 아니지만 내 가족과 동일한 권한을 가진 일원으로 인정받은 사람



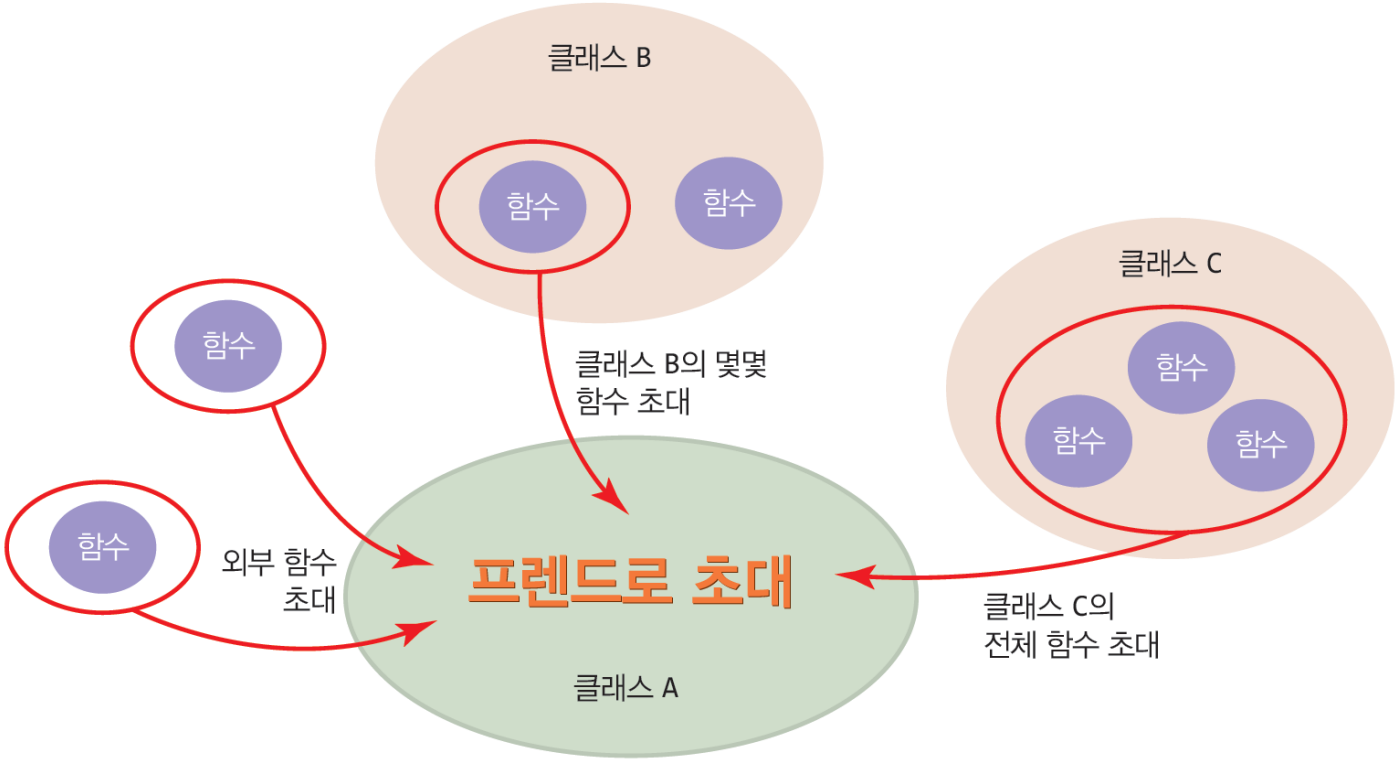
C++ 프렌드

- 프렌드 함수
  - 클래스의 멤버 함수가 아닌 외부 함수
    - 전역 함수
    - 다른 클래스의 멤버 함수
  - friend 키워드로 클래스 내에 선언된 함수
    - 클래스의 모든 멤버를 접근할 수 있는 권한 부여
    - 프렌드 함수라고 부름
  - 프렌드 선언의 필요성
    - 클래스의 멤버로 선언하기에는 무리가 있고, 클래스의 모든 멤버를 자유롭게 접근할 수 있는 일부 외부 함수 작성 시

항목	세상의 친구	프렌드 함수
존재	가족이 아님. 외부인	클래스 외부에 작성된 함수. 멤버가 아님
자격	가족의 구성원으로 인정받음. 가족의 모든 살림살이에 접근 허용	클래스의 멤버 자격 부여. 클래스의 모든 멤버에 대해 접근 가능
선언	친구라고 소개	클래스 내에 friend 키워드로 선언
개수	친구의 명수에 제한 없음	프렌드 함수 개수에 제한 없음

프렌드로 초대하는 3 가지 유형

- 프렌드 함수가 되는 3 가지
  - 전역 함수 : 클래스 외부에 선언된 전역 함수
  - 다른 클래스의 멤버 함수 : 다른 클래스의 특정 멤버 함수
  - 다른 클래스 전체 : 다른 클래스의 모든 멤버 함수



클래스의 friend 선언

```
class Boy
{
private:
    int height;
    friend class Girl;
public:
    Boy(int len) : height(len)
    { }
    . . . . .
};
```

Girl 클래스에 대한 friend 선언!

```
class Girl
{
private:
    char phNum[20];
public:
    Girl(char * num)
    {
        strcpy(phNum, num);
    }
    void ShowYourFriendInfo(Boy &frn)
    {
        cout<<"His height: "<<frn.height<<endl;
    }
};
```

Girl이 Boy의 friend로 선언되었으므로, private 멤버에 직접접근 가능

- friend 선언은 private 멤버의 접근을 허용
- friend 선언은 정보은닉에 반하는 선언이기 때문에 매우 제한적으로 선언되어야 한다.

Friend 선언 예제

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

class Girl;

class Boy
{
private:
	int height;
	friend class Girl;
public:
	Boy(int len) : height(len)
	{ }
	void ShowYourFriendInfo(Girl &frn);
};

class Girl
{
private:
	char phNum[20];
public:
	Girl(char * num)
	{
		strcpy(phNum, num);
	}
	void ShowYourFriendInfo(Boy &frn);
	friend class Boy;
};
```

```
void Boy::ShowYourFriendInfo(Girl &frn)
{
	cout<<"Her phone number: "<<frn.phNum<<endl;
}

void Girl::ShowYourFriendInfo(Boy &frn)
{
	cout<<"His height: "<<frn.height<<endl;
}

int main(void)
{
	Boy boy(170);
	Girl girl("010-1234-5678");

	boy.ShowYourFriendInfo(girl);
	girl.ShowYourFriendInfo(boy);
	return 0;
}
```

함수의 friend 선언

```
class Point
{
private:
    int x;
    int y;
public:
    Point(const int &xpos, const int &ypos) : x(xpos), y(ypos)
    { }
    friend Point PointOP::PointAdd(const Point&, const Point&);
    friend Point PointOP::PointSub(const Point&, const Point&);
    friend void ShowPointPos(const Point&);
};
```

전역변수 대상의 friend 선언

이렇듯 클래스의 특정 멤버함수를 대상으로도 friend 선언이 가능하다.

```
Point PointOP::PointAdd(const Point& pnt1, const Point& pnt2)
{
    opcnt++;
    return Point(pnt1.x+pnt2.x, pnt1.y+pnt2.y);
}
Point PointOP::PointSub(const Point& pnt1, const Point& pnt2)
{
    opcnt++;
    return Point(pnt1.x-pnt2.x, pnt1.y-pnt2.y);
}
```

private 멤버 접근

private 멤버 접근

```
void ShowPointPos(const Point& pos)
{
    cout<<"x: "<<pos.x<<", ";
    cout<<"y: "<<pos.y<<endl;
}
```

private 멤버 접근

MyFriendFunction.cpp

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

class Point;

class PointOP
{
private:
	int opcnt;
public:
	PointOP() : opcnt(0)
	{ }

	Point PointAdd(const Point&, const Point&);
	Point PointSub(const Point&, const Point&);
	~PointOP()
	{
		cout<<"Operation times: "<<opcnt<<endl;
	}
};

class Point
{
private:
	int x;
	int y;
public:
	Point(const int &xpos, const int &ypos) : x(xpos), y(ypos)
	{ }
	friend Point PointOP::PointAdd(const Point&, const Point&);
	friend Point PointOP::PointSub(const Point&, const Point&);
	friend void ShowPointPos(const Point&);
};
```

```
Point PointOP::PointAdd(const Point& pnt1, const Point& pnt2)
{
	opcnt++;
	return Point(pnt1.x+pnt2.x, pnt1.y+pnt2.y);
}

Point PointOP::PointSub(const Point& pnt1, const Point& pnt2)
{
	opcnt++;
	return Point(pnt1.x-pnt2.x, pnt1.y-pnt2.y);
}

int main(void)
{
	Point pos1(1, 2);
	Point pos2(2, 4);
	PointOP op;

	ShowPointPos(op.PointAdd(pos1, pos2));
	ShowPointPos(op.PointSub(pos2, pos1));
	return 0;
}

void ShowPointPos(const Point& pos)
{
	cout<<"x: "<<pos.x<<", ";
	cout<<"y: "<<pos.y<<endl;
}
```

Chapter 05-3. static

정적멤버(static member)

- 정적멤버
  - 같은 클래스 형으로 선언된 여러 객체들이 동일한 하나의 자료를 공유
  - 클래스의 정의 시 멤버를 static이란 키워드로 정의
  - 클래스 내에 선언된 정적멤버는 정적멤버가 선언된 클래스로 사용 영역이 제한된 전역변수(global variable)
  - 클래스 내에서 정적멤버를 선언할 때 정적 멤버 자체가 정의되는 것은 아니기 때문에 클래스 밖에서 정적멤버를 정의하는 선언 필요
  - 모든 정적멤버 변수는 특별히 초기값을 명시하지 않는 한 0으로 초기화

정적멤버(static member)

- Static 멤버의 등장
  - 전역 변수와 전역 함수를 일부 대처하기 위해 등장

- Static 키워드의 효과
  - 모든객체가 공유할 수 있는 멤버

정적멤버(static member)

- static 멤버의 특징
  - 클래스 변수, 클래스 함수라 한다.
  - main 함수 호출 이전에 메모리 공간에 올라가서 초기화 (전역변수와 동일)
  - 선언된 클래스의 객체 내에 직접 접근 허용
  - static 멤버 초기화문으로 초기화해야 함

static 멤버와 non-static 멤버의 특성

- 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

전역변수가 필요한 상황

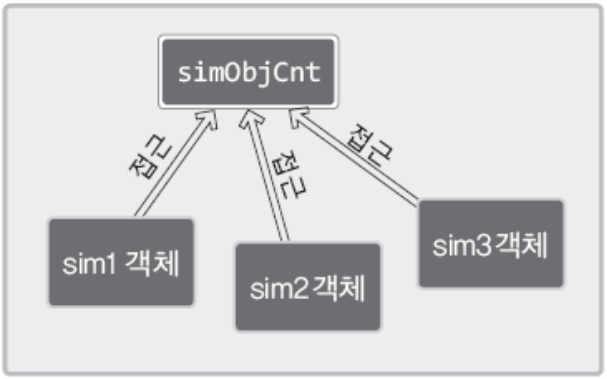
→ 객체지향이기때문에 전역변수를 사용할 것을 권하지 않음

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

int simObjCnt=0;
int cmxObjCnt=0;

class SoSimple
{
public:
    SoSimple()
    {
        simObjCnt++;
        cout<<simObjCnt<<"번째 SoSimple 객체
    "<<endl;
    }
};

class SoComplex
{
public:
    SoComplex()
    {
        cmxObjCnt++;
        cout<<cmxObjCnt<<"번째 SoComplex 객체
    "<<endl;
    }
    SoComplex(SoComplex &copy)
    {
        cmxObjCnt++;
        cout<<cmxObjCnt<<"번째 SoComplex 객체
    "<<endl;
    }
};

int main(void)
{
    SoSimple sim1;
    SoSimple sim2;

    SoComplex com1;
    SoComplex com2=com1;
    SoComplex();
    return 0;
}
```

static 멤버변수(클래스 변수)

```
class SoSimple
{
private:
    static int simObjCnt;    // static 멤버변수, 클래스 변수
public:
    SoSimple()
    {
        simObjCnt++;
        cout<<simObjCnt<<"번째 SoSimple 객체"<<endl;
    }
};

int SoSimple::simObjCnt=0;    // static 멤버변수의 초기화
```

```
int main(void)
{
    SoSimple sim1;
    SoSimple sim2;
    SoSimple sim3;
    . . . .
}
```



static 변수는 객체 별로 존재하는 변수가 아닌, 프로그램 전체 영역에서 하나만 존재하는 변수이다.

프로그램 실행과 동시에 초기화되어 메모리 공간에 할당된다.

StaticMember.cpp

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

class SoSimple
{
private:
    static int simObjCnt;
public:
    SoSimple()
    {
        simObjCnt++;
        cout<<simObjCnt<<"번째 SoSimple 객체
    "<<endl;
    }
};
int SoSimple::simObjCnt=0;

class SoComplex
{
private:
    static int cmxObjCnt;
public:
    SoComplex()
    {
        cmxObjCnt++;
        cout<<cmxObjCnt<<"번째 SoComplex 객체
    "<<endl;
    }
    SoComplex(SoComplex &copy)
    {
        cmxObjCnt++;
        cout<<cmxObjCnt<<"번째 SoComplex 객체
    "<<endl;
    }
};
int SoComplex::cmxObjCnt=0;

int main(void)
{
    SoSimple sim1;
    SoSimple sim2;

    SoComplex cmx1;
    SoComplex cmx2=cmx1;
    SoComplex();
    return 0;
}
```

C언어에서 이야기한 static

- 전역변수에 선언된 static의 의미
  - ➡ 선언된 파일 내에서만 참조를 허용하겠다는 의미
- 함수 내에 선언된 static의 의미
  - ➡ 한번만 초기화되고, 지역변수와 달리 함수를 빠져나가도 소멸되지 않는다.

```
void Counter()
{
    static int cnt;
    cnt++;
    cout<<"Current cnt: "<<cnt<<endl;
}

int main(void)
{
    for(int i=0; i<10; i++)
        Counter();
    return 0;
}
```

실행결과

```
Current cnt: 1
Current cnt: 2
Current cnt: 3
Current cnt: 4
Current cnt: 5
Current cnt: 6
Current cnt: 7
Current cnt: 8
Current cnt: 9
Current cnt: 10
```

static 멤버 사용 : 객체의 멤버로 접근

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

```
객체.static멤버
객체포인터->static멤버
```

  - 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
```

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

class Person {
public:
    double 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;
}
```

- 출력결과는 ?

- 각 변수값들이 어떻게 변경될까 생각해봅시다.

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

class Person {
public:
    double 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

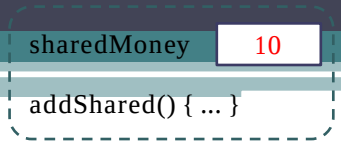
```
#include <iostream>
using namespace std;

class Person {
public:
    double 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

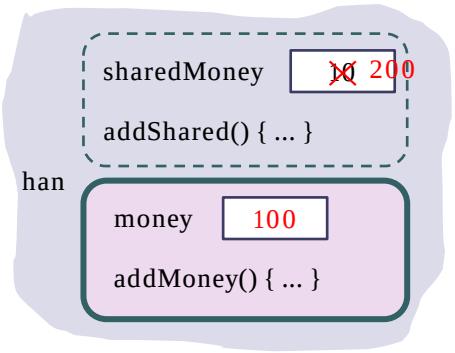
han과 lee의 money는 각각 100, 350

han과 lee의 sharedMoney는 공통 400

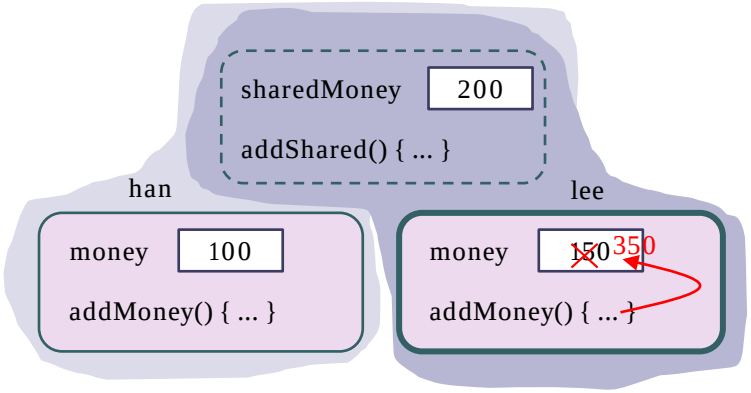
main()이 시작하기 직전



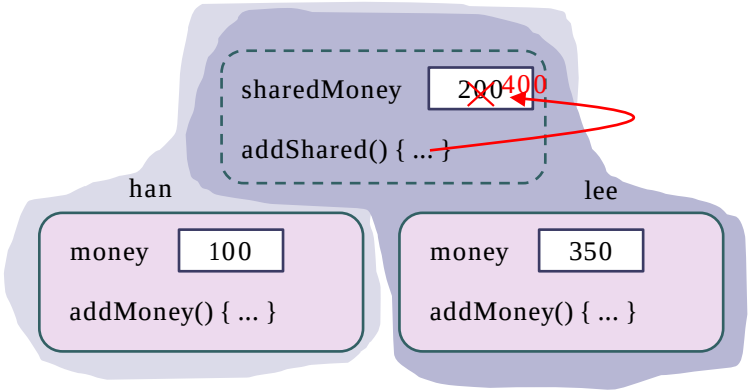
Person han;
han.money = 100;
han.sharedMoney = 200;



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



lee.addshared(200);



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

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

클래스명::static멤버

```
han.sharedMoney = 200;          <->     Person::sharedMoney = 200;
lee.addShared(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:
    double 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;
}
```

- 출력결과는 ?

- 각 변수값들이 어떻게 변경될까 생각해봅시다.

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

class Person {
public:
    double 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;
}
```

han 객체가 생기기전부터 static 멤버 접근

```
60
100 400
```

sharedMoney 400

han의 money 100

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

class Person {
public:
    double 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;
}
```

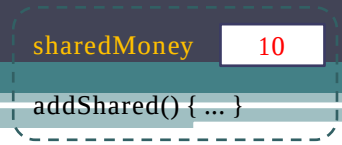
han 객체가 생기기전부터 static 멤버 접근

```
60
100 400
```

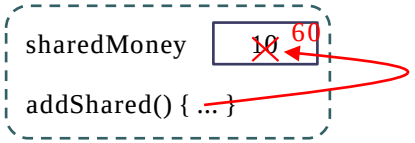
sharedMoney 400

han의 money 100

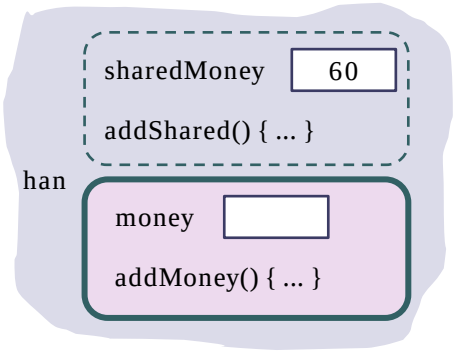
main()이 시작하기 직전



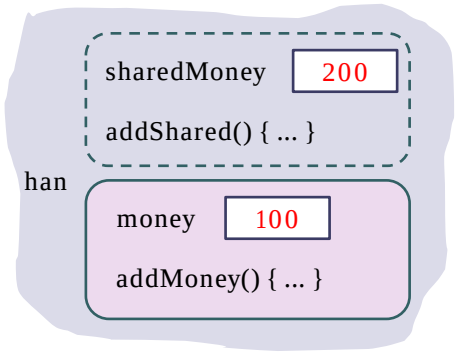
Person::addShared(50);



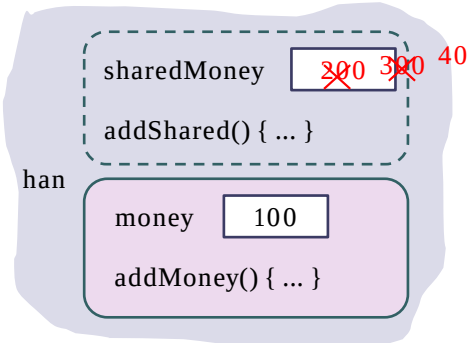
Person han;



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



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



참고문헌

- 뇌를 자극하는 C++ 프로그래밍, 이현창, 한빛미디어, 2011
- 열혈 C++ 프로그래밍(개정판), 윤성우, 오렌지미디어, 2012
- C++ ESPRESSO, 천인국 저, 인피니티북스, 2011
- 명품 C++ Programming, 황기태 , 생능출판사, 2013

Q & A

추가 자료

C의 const

```
const double PI=3.14;
PI=3.1415;
                                    // 컴파일 오류
```

```
const int val;
val=20;
                                    // 컴파일 오류
```

C의 const

```
int n=10;
const int* pN=&n;
*pN=20;    // 컴파일 오류
```

```
int n1=10;
int n2=20;
int* const pN=&n1;
*pN=20;
pN=&n2;    //컴파일 오류
```

멤버변수의 상수화

```
#include<iostream>
using std::cout;
using std::endl;

class Student
{
	const int id;
	int age;
	char name[20];
	char major[30];
public:
	Student(int _id, int _age, char* _name, char* _major)
	{
		id=_id;	//error
		age=_age;
		strcpy(name, _name);
		strcpy(major, _major);
	}

	void ShowData()
	{
		cout<<"이름: "<<name<<endl;
		cout<<"나이: "<<age<<endl;
		cout<<"학번: "<<id<<endl;
		cout<<"학과: "<<major<<endl;
	}
};
```

```
int main()
{
	Student Kim(200577065, 20, "Kim Gil Dong",
                                        "Computer Eng.");
	Student Hong(200512065, 19, "Hong Gil Dong",
                                        "Electronics Eng.");

	Kim.ShowData();
	cout<<endl;
	Hong.ShowData();

	return 0;
}
```

- 컴파일 에러
  - 상수값을 생성자에서 초기화하여 발생
- Member initializer 사용
  - Const 멤버변수 초기화

멤버변수의 상수화

```
#include<iostream>
using std::cout;
using std::endl;

class Student
{
	const int id;
	int age;
	char name[20];
	char major[30];
public:
	Student(int _id, int _age, char* _name, char* _major) : id(_id), age(_age)
	{
		strcpy(name, _name);
		strcpy(major, _major);
	}

	void ShowData()
	{
		cout<<"이름: "<<name<<endl;
		cout<<"나이: "<<age<<endl;
		cout<<"학번: "<<id<<endl;
		cout<<"학과: "<<major<<endl;
	}
};
```

❖Initializer는
생성자 함수 호출전에 초기화됨.

```
int main()
{
	Student Kim(200577065, 20, "Kim Gil Dong",
"Computer Eng.");
	Student Hong(200512065, 19, "Hong Gil Dong",
"Electronics Eng.");

	Kim.ShowData();
	cout<<endl;
	Hong.ShowData();

	return 0;
}
```

const 멤버 함수 예

```
#include <iostream>
using std::cout;
using std::endl;

class Count
{
	int cnt;
public :
	Count() : cnt(0){}
	const int* GetPtr() const {
		return &cnt;
	}

	void Increment(){
		cnt++;
	}

	void ShowData() const {
		ShowIntro();
		cout<<cnt<<endl;
	}
	void ShowIntro() const {
		cout<<"현재 count의 값 : "<<endl;
	}
};
```

```
int main()
{
	Count count;
	count.Increment();
	count.Increment();
	count.ShowData();

	return 0;
}
```

static 멤버변수의 접근방법

```
class SoSimple
{
public:
    static int simObjCnt;
public:
    SoSimple()
    {
        simObjCnt++;
    }
};
int SoSimple::simObjCnt=0;
```

접근 case 1

```
int main(void)
{
    cout<<SoSimple::simObjCnt<<"번째 SoSimple 객체"<<endl;
    SoSimple sim1;
    SoSimple sim2;
    cout<<SoSimple::simObjCnt<<"번째 SoSimple 객체"<<endl;
    cout<<sim1.simObjCnt<<"번째 SoSimple 객체"<<endl;
    cout<<sim2.simObjCnt<<"번째 SoSimple 객체"<<endl;
    return 0;
}
```

접근 case 2

접근 case 3

static 변수가 선언된 외부에서의 접근이 가능 하려면,
해당 변수가 public으로 선언되어야 한다.

실행결과

```
0번째 SoSimple 객체
2번째 SoSimple 객체
2번째 SoSimple 객체
2번째 SoSimple 객체
```

PublicStaticMember

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

class SoSimple
{
public:
	static int simObjCnt;
public:
	SoSimple()
	{
		simObjCnt++;
	}
};
int SoSimple::simObjCnt=0;

int main(void)
{
	cout<<SoSimple::simObjCnt<<"번째 SoSimple 객체"<<endl;
	SoSimple sim1;
	SoSimple sim2;

	cout<<SoSimple::simObjCnt<<"번째 SoSimple 객체"<<endl;
	cout<<sim1.simObjCnt<<"번째 SoSimple 객체"<<endl;
	cout<<sim2.simObjCnt<<"번째 SoSimple 객체"<<endl;
	return 0;
}
```

static 멤버함수

static 멤버변수의 특징과 일치한다.

- 선언된 클래스의 모든 객체가 공유한다.
- public으로 선언이 되면, 클래스의 이름을 이용해서 호출이 가능하다.
- 객체의 멤버로 존재하는 것이 아니다.

```
class SoSimple
{
private:
    int num1;
    static int num2;
public:
    SoSimple(int n): num1(n)
    { }
    static void Adder(int n)
    {
        num1+=n;     // 컴파일 에러 발생
        num2+=n;
    }
};
int SoSimple::num2=0;
```

static 함수는 객체 내에 존재하는 함수가 아니기 때문에 멤버변수나 멤버함수에 접근이 불가능하다.

static 함수는 static 변수에만 접근 가능하고, static 함수만 호출 가능하다.

const static 멤버와 mutable

```
class CountryArea
{
public:
    const static int RUSSIA    =1707540;
    const static int CANADA    =998467;
    const static int CHINA     =957290;
    const static int SOUTH_KOREA   =9922;
};

int main(void)
{
    cout<<"러시아 면적: "<<CountryArea::RUSSIA<<"㎢"<<endl;
    cout<<"캐나다 면적: "<<CountryArea::CANADA<<"㎢"<<endl;
    cout<<"중국 면적: "<<CountryArea::CHINA<<"㎢"<<endl;
    cout<<"한국 면적: "<<CountryArea::SOUTH_KOREA<<"㎢"<<endl;
    return 0;
}
```

const static 멤버변수는, 클래스가 정의될 때 지정된 값이 유지되는 상수이기 때문에, 위 예제에서 보이는 바와 같이 초기화가 가능하도록 문법으로 정의하고 있다.

```
class SoSimple
{
private:
    int num1;
    mutable int num2;
public:
    SoSimple(int n1, int n2)
        : num1(n1), num2(n2)
    { }
    void CopyToNum2() const
    {
        num2=num1;
    }
};
```

mutable로 선언된 멤버변수는 const 함수 내에서 값의 변경이 가능하다.

ConstStaticMember.cpp

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

class CountryArea
{
public:
    const static int RUSSIA          =1707540;
    const static int CANADA          =998467;
    const static int CHINA           =957290;
    const static int SOUTH_KOREA     =9922;
};

int main(void)
{
    cout<<"러시아 면적: "<<CountryArea::RUSSIA<<"㎢"<<endl;
    cout<<"캐나다 면적: "<<CountryArea::CANADA<<"㎢"<<endl;
    cout<<"중국 면적: "<<CountryArea::CHINA<<"㎢"<<endl;
    cout<<"한국 면적: "<<CountryArea::SOUTH_KOREA<<"㎢"<<endl;
    return 0;
}
```

Mutable.cpp

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

class SoSimple
{
private:
	int num1;
	mutable int num2;
public:
	SoSimple(int n1, int n2)
		: num1(n1), num2(n2)
	{ }
	void ShowSimpleData() const
	{
		cout<<num1<<", "<<num2<<endl;
	}
	void CopyToNum2() const
	{
		num2=num1;
	}
};

int main(void)
{
	SoSimple sm(1, 2);
	sm.ShowSimpleData();
	sm.CopyToNum2();
	sm.ShowSimpleData();
	return 0;
}
```

explicit & mutable

- explicit
  - 명시적 호출만 허용한다.

- mutable
  - const에 예외를 둔다

explicit & mutable

```
/*    explicit.cpp    */

#include<iostream>
using std::cout;
using std::endl;

class AAA
{
public:
    explicit AAA(int n){
                cout<<"explicit AAA(int n)"<<endl;
     }
};

int main(void)
{
     AAA a1=10;

     return 0;
}
```

explicit & mutable

```
/*    mutable.cpp    */
#include<iostream>
using std::cout;
using std::endl;

class AAA
{
private:
      mutable int val1;
      int val2;
public:
      void SetData(int a, int b) const
      {
            val1=a;  // val1이 mutable이므로 OK!
            val2=b;  // Error!
      }
};
```

```
int main(void)
{
      AAA a1;
      a1.SetData(10, 20);
      return 0;
}
```