235-237p

임시객체의 생성과 소멸

- **생성**
  - 변수로 대입해 놓지 않고 객체 생성하면 임시객체가 됨.
  - 함수 반환시 참조자 아닌 객체 반환할 때 생성됨.
    - ❖ 단, 이 때 함수가 반환하는 객체를 선언된 변수에 받으면 2번 연속 생성하지 않고 변수만 생성하여 반환객체를 보관.
- **소멸**
  - 임시 객체는 생성된 문장이 끝나면 바로 소멸됨.
  - 단, 참조자에 의해 참조된 임시객체는 참조자의 생명주기를 따라감.

임시객체의 생성과 소멸
(IKnowTempObj.cpp)

235-237p

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

class Temporary {
private:
    int num;
public:
    Temporary(int n) : num(n) {
        cout << "create obj: " << num << endl;
    }
    ~Temporary() {
        cout << "destroy obj: " << num << endl;
    }
    void ShowTempInfo() {
        cout << "My num is " << num << endl;
    }
};

int main(void) {
    Temporary t1(10);

    Temporary(100);
    cout << "********** after make1!" << endl << endl;

    Temporary(200).ShowTempInfo();
    cout << "********** after make2!" << endl << endl;

    const Temporary& ref = Temporary(300);
    cout << "********** end of main!" << endl << endl;
    return 0;
}
```

[결과]

```
create obj: 10
create obj: 100
destroy obj: 100
********** after make1!

create obj: 200
My num is 200
destroy obj: 200
********** after make2!

create obj: 300
********** end of main!

destroy obj: 300
destroy obj: 10
```

반환 시점에 생성되는 임시객체의 소멸
(ReturnObjDeadTime.cpp)

238-239p

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

class SoSimple {
    int num;
public:
    SoSimple(int n) : num(n) { cout << "New Object: " << this << endl; }
    SoSimple(const SoSimple& copy) : num(copy.num) {
        cout << "New Copy obj: " << this << endl;
    }
    ~SoSimple() { cout << "Destroy obj: " << this << endl; }
};

SoSimple SimpleFuncObj(SoSimple ob) {
    cout << "Parm ADR: " << &ob << endl;
    return ob;
}

int main(void) {
    SoSimple obj(7);
    SimpleFuncObj(obj);

    cout << endl;
    SoSimple tempRef = SimpleFuncObj(obj);
    cout << "Return Obj " << &tempRef << endl;
    return 0;
}
```

변수에 받지 않은 반환용 임시객체 즉시삭제

변수에 받은 반환용 임시객체 나중에 삭제

[결과]

```
New Object: 009DF9DC
New Copy obj: 009DF8C4
Parm ADR: 009DF8C4
New Copy obj: 009DF8F8
Destroy obj: 009DF8C4
Destroy obj: 009DF8F8

New Copy obj: 009DF8C4
Parm ADR: 009DF8C4
New Copy obj: 009DF9D0
Destroy obj: 009DF8C4
Return Obj 009DF9D0
Destroy obj: 009DF9D0
Destroy obj: 009DF9DC
```

7. friend, static, const

7주차

const 객체과 const 멤버 함수 (ConstObject.cpp)

244-245p

- const로 선언된 객체를 대상으로는 const 멤버함수만 호출 가능.
- const는 가능한 많이 사용할수록 좋다.

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

class SoSimple {
        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;
}
```

AddNum은 const함수가 아니므로 호출 불가.

245-247p

const 와 함수 overloading
(ConstOverloading.cpp)

◆ **const 선언 유무도 함수 overloading 의 조건이 됨.**

```
#include <iostream>
using namespace std;
class SoSimple {
        int num;
public:
        SoSimple(int n) : num(n) { }
        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;
}
```

const 객체에 대한 SimpleFunc 호출은 const 함수가 수행됨.

[결과]
SimpleFunc: 2
const SimpleFunc: 7
const SimpleFunc: 2
const SimpleFunc: 7

friend 클래스 선언
(MyFriendClass.cpp 1/2)

247-251p

- **내가 friend class 로 선언해 준 클래스에서는 나의 private 멤버를 사용할 수 있다. (정보은닉에 위배(멤버변수가 public인 것보다는 낫다!)되므로 최소한만 사용하자!)**

```
#define _CRT_SECURE_NO_WARNINGS
#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;
};
```

Girl 이라는 클래스가 있음을 알림.
friend class Girl;
이 이러한 의미도 포함하므로 여기서는 없어도 됨.

Friend 선언은 클래스 선언 내의 어느 위치, 어느 한정자 안에 있어도 같은 효력.

함수 구현부에서 2개 클래스 멤버를 모두 이용할 수 있도록 클래스 선언을 먼저 함.

247-251p

friend 클래스 선언
(MyFriendClass.cpp 2/2)

```
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((char *)"010-1234-5678");

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

friend 함수 선언
(MyFriendFunction.cpp 1/3)

◆ 클래스 멤버 함수 또는 전역함수를 friend로 선언. -> 해당 함수에서 나의 private 멤버에 접근 가능

```
#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;
        }
};
```

friend 함수 선언
(MyFriendFunction.cpp 2/3)

```
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);
}
```

다른 클래스의 멤버함수를 friend로 선언

전역 함수를 friend로 선언. 함수원형선언도 같이 되므로 별도의 원형선언은 필요 없음.

PointOP의 멤버이지만 Point 클래스의 private 멤버 사용가능.

friend 함수 선언
(MyFriendFunction.cpp 3/3)

```
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;
}
```

전역함수이지만 Point 클래스의 private 멤버 사용가능.

254-255p

C언어에서의 static

◆ C언어에서의 static

- static 전역변수
  - ❖ 선언된 파일 안에서만 접근 가능
- static 지역변수
  - ❖ 1회만 초기화됨. 접근 영역은 지역, 생명주기는 전역.

전역변수 사용
(NeedGlobal.cpp 1/2)

◆ 2개 클래스 각각의 객체가 몇 개 씩 생성되는지 카운트

```
#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;
        }
}
```

전역변수이므로 다른 클래스에서 접근해도 막지 못함.

256-257p

전역변수 사용
(NeedGlobal.cpp 2/2)

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

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

클래스의 static 멤버변수

- Static 멤버변수
  - 클래스 별로 1개씩 (객체별로가 아님!) 전역변수를 갖는 것.
  - 생명주기는 전역변수와 같지만 접근 방법은 클래스 멤버의 규칙을 따름.
  - 객체와 연결된 것이 아니고 클래스와 연결된 것이므로 클래스 이름을 사용하여 접근하는 것이 좋음.
    - cout << SoSimple::count; ← 올바른 접근!
    - SoSimple obj1;
      cout << obj1::count; ← 가능은 하지만 자제!

클래스의 static 멤버변수
(StaticMember.cpp 1/3)

257-261p

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

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

클래스 멤버이므로 다른 클래스에서 실수로 접근할 수 없음.

Static 멤버 변수는 클래스 외부에서 초기화

클래스의 static 멤버변수
(StaticMember.cpp 2/3)

257-261p

```
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;
```

클래스 멤버이므로 다른 클래스에서 실수로 접근할 수 없음.

Static 멤버 변수는 클래스 외부에서 초기화

257-261p

클래스의 static 멤버변수
(StaticMember.cpp 3/3)

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

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

클래스 이름을 통한 static 멤버변수 접근
(PublicStaticMember.cpp)

```
#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;
}
```

public static 멤버에 외부에서 접근할 때는 클래스 이름을 통하는 것이 좋음. (객체 이름을 통하지 말고)

객체가 없어도 static멤버는 이미 있다.