SEOIL UNIVERSITY

C++ 스마트 포인터 1

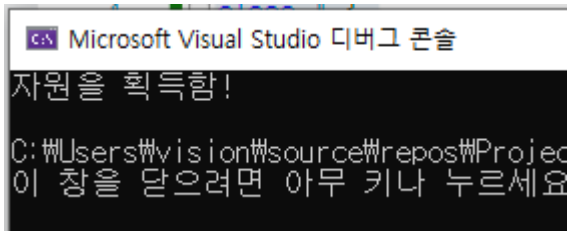
자원(resource) 관리

SEOIL UNIVERSITY

```
class A {
    int* data;
public:
    A() {
        data = new int[100];
        cout << "자원을 획득함!"
<< endl;
    }
    ~A() {
        cout << "소멸자 호출!" << endl;
        delete[] data;
    }
};


int main() {
    A* pa = new A();

     return 0;
}
```



// 할당된 객체가 소멸되지 않음!
// 400 바이트 (4 * 100) 만큼의 메모리 누수 발생

자원(resource) 관리

SEOIL UNIVERSITY

delete pa;

delete 는 메모리를 해제하기 직전 가리키는 객체의 소멸자를 호출
프로그램의 크기가 커지면, 자원을 해제하는 위치가 애매한 경우
프로그래머 책임

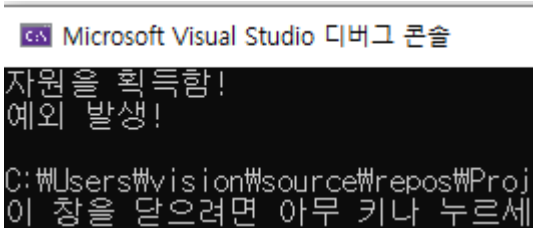
자원(resource) 관리

SEOIL UNIVERSITY

```
class A {
    int* data;
public:
    A() {
        data = new int[100];
        cout << "자원을 획득함!" << endl;
    }
    ~A() {
        cout << "자원을 해제함!" << endl;
        delete[] data;
    }
};


int main() {

    try {
        A* pa = new A();
        throw 1;
        delete pa;
    }
    catch (int i) {
        cout << "예외 발생!" << endl;
    }
    return 0;
}
```



스마트 포인터(smart pointer)

SEOIL UNIVERSITY

- 바로 흔히 RAII 라 불리는 자원의 획득 초기화
- 똑똑하게 작동하는 포인터 객체를 스마트 포인터(smart pointer)
- C++ 11 에서는 auto_ptr를 보완한
- unique_ptr , shared_ptr

스마트 포인터(smart pointer) : unique_ptr

SEOIL UNIVERSITY

- 객체의 유일한 소유권 - unique_ptr
- 1. 메모리를 해제하지 않는 문제
- 2. 해제된 메모리를 다시 참조하는 경우

```
Data* data = new Data();
Date* data2 = data;
// data 의 입장 : 사용 다 했으니 소멸시켜야지.
delete data;
// ...
// data2 의 입장 : 나도 사용 다 했으니 소멸시켜야지
delete data2;
```

- data2 가 이미 소멸된 객체를 다시 소멸
- 결과는 메모리 오류가 나면서 프로그램이 죽음
- 소멸된 객체를 다시 소멸시켜서 발생하는 버그 double free 버그

스마트 포인터(smart pointer)

SEOIL UNIVERSITY

- 객체의 소유권이 명확하지 않아서 발생하는 문제
- 포인터에 객체의 유일한 소유권을 부여
- "이 포인터 말고는 객체를 소멸시킬 수 없다"
- data 에 new Data() 로 생성된 객체의 소유권을 보유한다면
- delete data 만 가능하고, delete data2 는 불가능
- C++ 에서는 이렇게, 특정 객체에 유일한 소유권을 부여하는 포인터 객체 unique_ptr

스마트 포인터(smart pointer)

SEOIL UNIVERSITY

```
class A {
    int* data;
public:
    A() {
        data = new int[100];
        cout << "자원을 획득함!" << endl;
    }
    ~A() {
        cout << "자원을 해제함!" << endl;
        delete[] data;
    }

    void view() {
        cout << "일반 포인터와 동일하게 사용가능!" << endl;
    }

};

int main() {
    unique_ptr<A> pa(new A());
    pa->view();
    return 0;
}
```

스마트 포인터(smart pointer)

- **unique_ptr 을 정의하는 부분**
  - std::unique_ptr<A> pa(new A());
- **unique_ptr 를 정의하기 위해서는 템플릿에 인자로, 포인터가 가리킬 클래스를 전달**
- **pa 는 A 클래스의 객체를 가리키는 포인터**
  - A* pa = new A(); // 같은 상황
  - pa->view();
- **unique_ptr 은 -> 연산자를 오버로드해서 마치 포인터를 다루는 것과 같이 사용**

스마트 포인터(smart pointer)

SEOIL UNIVERSITY

```
class A {
int* data;
public:
A() {
cout << "자원을 획득함!" << endl;
data = new int[100];
}
void view() {
cout << "일반 포인터와 동일하게 사용가능!" << endl;
}
~A() {
cout << "자원을 해제함!" << endl;
delete[] data;
}
};

int main() {
unique_ptr<A> pa(new A());
// pb 도 객체를 가리키게 할 수 있을까?
unique_ptr<A> pb = pa;
}
```

컴파일 오류

스마트 포인터(smart pointer)

SEOIL UNIVERSITY

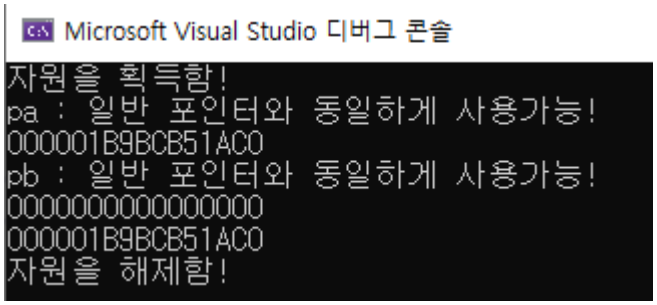
```
class A {
int* data;
public:
A() {
cout << "자원을 획득함!" << endl;
data = new int[100];
}
~A() {
cout << "자원을 해제함!" << endl;
delete[] data;
}
void view() {
cout << "일반 포인터와 동일하게 사용가능!" << endl;
}
};

int main() {
unique_ptr<A> pa(new A());
cout << "pa : ";
pa->view();
// pb 에 소유권을 이전.
unique_ptr<A> pb = move(pa);
cout << "pb : ";
pb->view();

return 0;
}
```

스마트 포인터(smart pointer)

- **pa가 가르키는 주소 확인**
  - 0 (nullptr)
- **cout << pa.get() << endl;**
- **pa 와 pb 모두 확인하면 서로 같은 주소 확인할것**



감사합니다.