Prog

#9. 상속과 포함

C++ Programming Tutor

9주차 수업 안내

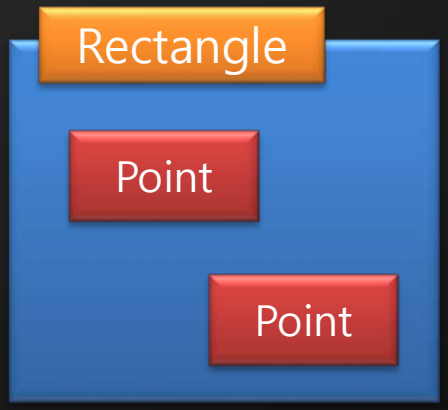
- 상속과 포함
  - 포함
    - 왜 포함을 사용해야 할까
    - 객체를 멤버로 갖는 클래스
  - 상속
    - 문서 저장 클래스
    - HTML 문서 저장 클래스
    - 클래스간의 형변환
    - 접근 제어

왜 포함을 사용해야 할까

포함 관계는 어떤 객체가 다른 객체를 포함하고 있는 관계!

Point

Point 클래스



왜 포함을 사용해야 할까

Prog

포함 관계의 장점?

재사용의 관점에서 바라보면...

기존에 만들어진 클래스를 사용해서 만들면 시간 절약!

또한 중복된 코드를 새롭게 만들지 않고!

객체를 멤버로 갖는 클래스

Prog

진짜 만들어 봅시다!

점 클래스를 사용해서 사각형 클래스 만들기!

객체를 멤버로 갖는 클래스

Prog

1. Point.h

```
#ifndef POINT_H
#define POINT_H

// Point 클래스 정의
class Point {
public:
    // 멤버 함수
    void Print() const;
    void Offset(int x_delta, int y_delta);
    void Offset(const Point& pt);
```

객체를 멤버로 갖는 클래스

Prog

1. Point.h

```
// 생성자들
Point();
Point(int initialX, int initialY);
Point(const Point& pt);

// 접근자
void SetX(int value);
void SetY(int value);
int GetX() const { return x; }
int GetY() const { return y; }
```

객체를 멤버로 갖는 클래스

Prog

1. Point.h

```
private:
    // 멤버 변수
    int x, y;
};

inline void Point::SetX(int value) {
    if (value < 0)          x = 0;
    else if (value > 100) x = 100;
    else                        x = value;
}
```

객체를 멤버로 갖는 클래스

Prog

1. Point.h

```
inline void Point::SetY(int value) {
    if (value < 0)          y = 0;
    else if (value > 100) y = 100;
    else                        y = value;
}

#endif
```

객체를 멤버로 갖는 클래스

Prog

2. Point.cpp

```
#include “Point.h”
#include <iostream>

void Point::Offset(int x_delta, int y_delta) {
    SetX(x + x_delta);
    SetY(y + y_delta);
}

void Point::Offset(const Point& pt) {
    Offset(pt.x, pt.y);
}
```

객체를 멤버로 갖는 클래스

Prog

2. Point.cpp

```
Point::Point(const Point& pt) {
    x = pt.x;
    y = pt.y;
}

Point::Point(int initialX, int initialY) {
    SetX(initialX);
    SetY(initialY);
}
```

객체를 멤버로 갖는 클래스

Prog

2. Point.cpp

```
Point::Point() {
    x = 0;
    y = 0;
}

void Point::Print() const {
    std::cout << "(" << x << ", "
        << y << ")₩n";
}
```

객체를 멤버로 갖는 클래스

Prog

3. Rect.h

```
#ifndef RECT_H
#define RECT_H

#include "Point.h"

class Rect {
public:
    // 생성자
    Rect();
```

객체를 멤버로 갖는 클래스

Prog

3. Rect.h

```
// 각 점의 값 지정 및 얻기
void SetTopLeft(const Point& topLeft);
void SetBottomRight(const Point& bottomRight);
void SetRect(int left, int top, int right, int bottom);
Point GetTopLeft() const;
Point GetBottomRight() const;
void GetRect(int& rect, int& top, int& right, int& bottom);

// 넓이, 높이 계산
int GetWidth() const;
int GetHeight() const;
```

객체를 멤버로 갖는 클래스

Prog

3. Rect.h

```
    // 내용 출력
    void Print() const;

protected:
    Point _topLeft;
    Point _bottomRight;
};

#endif
```

객체를 멤버로 갖는 클래스

Prog

4. Rect.cpp

```
#include “Rect.h”
#include <iostream>

Rect::Rect() { }

void Rect::SetTopLeft(const Point& topLeft) {
    _topLeft = topLeft;
}

void Rect::SetBottomRight(const Point& bottomRight) {
    _bottomRight = bottomRight;
}
```

객체를 멤버로 갖는 클래스

Prog

4. Rect.cpp

```
void Rect::SetRect(int left, int top, int right, int bottom) {
    _topLeft.SetX(left);
    _topLeft.SetY(top);
    _bottomRight.SetX(right);
    _bottomRight.SetY(bottom);
}

Point Rect::GetTopLeft() const {
    return _topLeft;
}
```

객체를 멤버로 갖는 클래스

Prog

4. Rect.cpp

```
Point Rect::GetBottomRight() const {
    return _bottomRight;
}

void Rect::GetRect(int& left, int& top,
    int& right, int& bottom) {
    left = _topLeft.GetX();
    top = _topLeft.GetY();
    right = _bottomRight.GetX();
    bottom = _bottomRight.GetY();
}
```

객체를 멤버로 갖는 클래스

Prog

4. Rect.cpp

```
// 넓이 계산
int Rect::GetWidth() const {
    return (_bottomRight.GetX() - _topLeft.GetX() + 1);
}

// 높이 계산
int Rect::GetHeight() const {
    return (_bottomRight.GetY() - _topLeft.GetY() + 1);
}
```

객체를 멤버로 갖는 클래스

Prog

4. Rect.cpp

```
// 내용 출력
void Rect::Print() const {
    std::cout << "{L = " << _topLeft.GetX()
        << ", T = " << _topLeft.GetY() << ", R = "
        << _bottomRight.GetX() << ", B = "
        << _bottomRight.GetY() << "}₩n";
}
```

객체를 멤버로 갖는 클래스

Prog

5. Example.cpp

```
#include “Rect.h”
#include <iostream>

int main() {
    // Rect 객체 생성
    Rect rc1;
    // 내용 출력
    rc1. Print();
    // 값을 변경
    rc1.SetRect(10, 20, 30, 40);
    // 내용 출력
    rc1.Print();
```

객체를 멤버로 갖는 클래스

Prog

5. Example.cpp

```
    // 값을 변경
    rc1.SetTopLeft(Point(20, 20));
    // 내용 출력
    rc1.Print();
    // 넓이, 높이 출력
    std::cout << "rc1.GetWidth() = "
        << rc1.GetWidth() << "₩n";
    std::cout << "rc1.GetHeight = "
        << rc1.GetHeight() << "₩n";

    return 0;
}
```

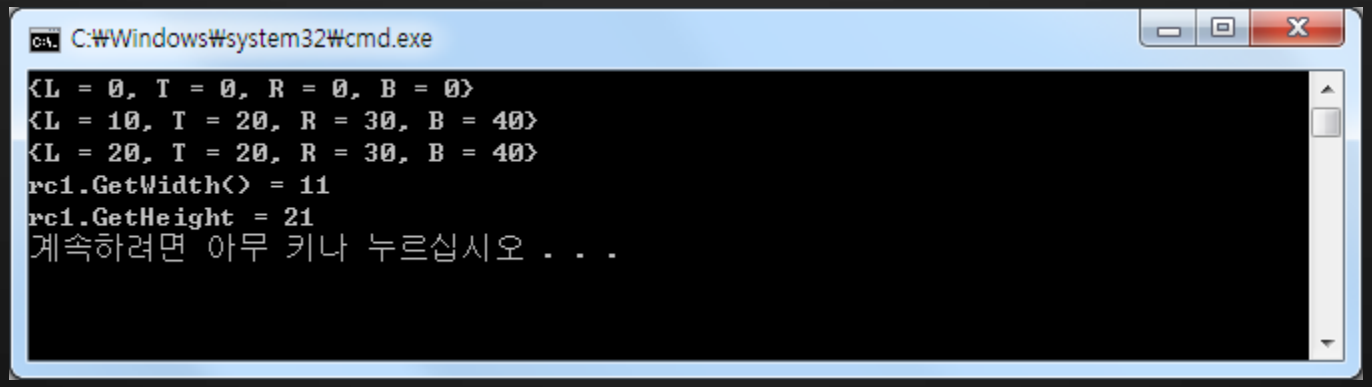
객체를 멤버로 갖는 클래스

Prog

객체를 멤버로 갖는 클래스

Prog

어쨌든 결과는 나온다!



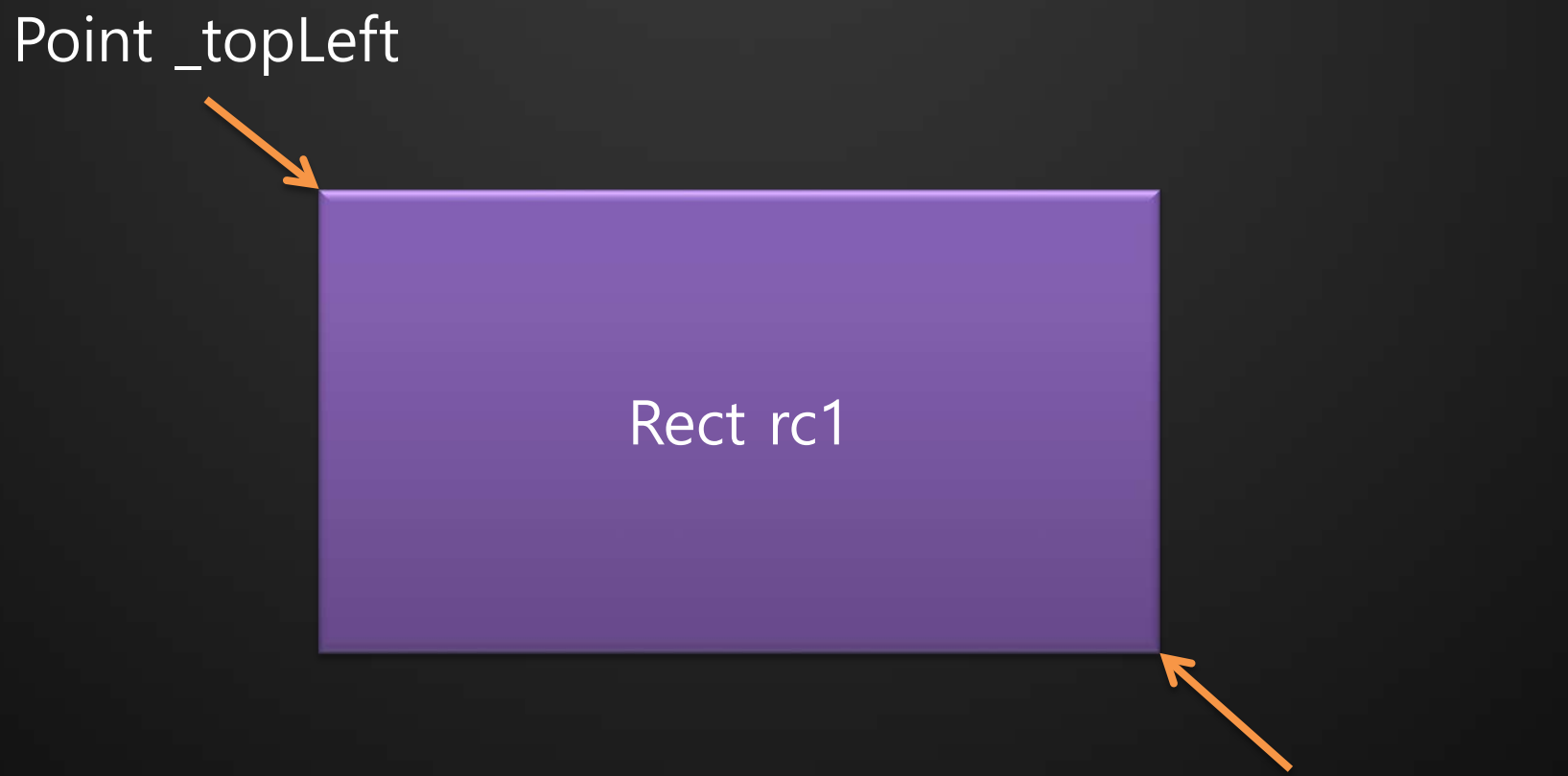
객체를 멤버로 갖는 클래스

Prog

그런데 우리가 뭘 만든거죠?

객체를 멤버로 갖는 클래스

Prog



객체를 멤버로 갖는 클래스

Prog

Q : Rect 객체의 위치가 무조건 0으로 초기화되네요

왜 그렇죠? 저는 Rect 객체의 위치를 처음에 지정하고 싶은데...

객체를 멤버로 갖는 클래스

Prog

A : 그건 Point 객체의 디폴트 생성자를 사용하기 때문이에요

Rect 객체의 위치를 처음에 지정하고 싶다면

Point 객체가 다른 생성자를 사용하게 하면 되겠죠!

객체를 멤버로 갖는 클래스

Prog

1. Rect.h

```
// 생성자
Rect();
Rect(const Point& topLeft, const Point& bottomRight);
Rect(int left, int top, int right, int bottom);
```

객체를 멤버로 갖는 클래스

Prog

2. Rect.cpp

```
Rect::Rect(const Point& topLeft,
    const Point& bottomRight)
    : _topLeft(topLeft), _bottomRight(bottomRight) { }

Rect::Rect(int left, int top, int right, int bottom)
: _topLeft(left, top), _bottomRight(right, bottom) { }
```

객체를 멤버로 갖는 클래스

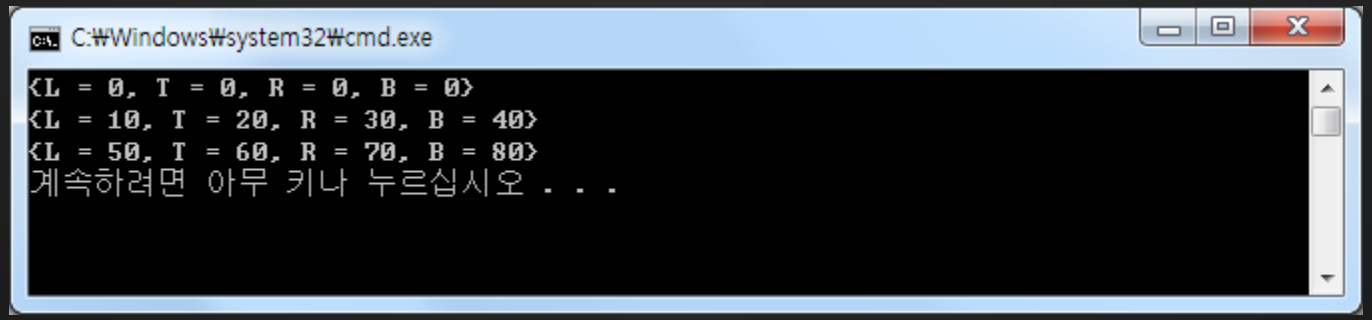
Prog

3. Example.cpp

```
int main() {
    Rect rc1;
    Rect rc2(Point(10, 20), Point(30, 40));
    Rect rc3(50, 60, 70, 80);

    rc1.Print();
    rc2.Print();
    rc3.Print();

    return 0;
}
```

객체를 멤버로 갖는 클래스

Prog



객체를 멤버로 갖는 클래스

Prog

지난 시간에 배운 생성자의 초기화 리스트 사용 용도?

객체를 멤버로 갖는 클래스

Prog

지난 시간에 배운 const 속성을 가지고 있는 멤버 변수나
레퍼런스 타입의 멤버 변수를 초기화할 때 사용

객체를 멤버로 갖는 클래스

Prog

하지만

멤버 변수인 객체의 생성자를 호출하는 데도 사용!

객체를 멤버로 갖는 클래스

Prog

```
Rect::Rect(int left, int top, int right, int bottom)
: _topLeft(left, top), _bottomRight(right, bottom) { }
```

_topLeft 객체의
생성자를 호출한다

_bottomRight 객체의
생성자를 호출한다

객체를 멤버로 갖는 클래스

Prog

이전에 초기화 리스트를 다루면서...

생성자보다 초기화 리스트가 먼저 실행된다는 걸 배웠다

또한 Point 객체의 생성자는 초기화 리스트에서 호출된다

객체를 멤버로 갖는 클래스

Prog

따라서

Point 객체들의 생성자가 Rect 객체의 생성자보다

먼저 실행된다는 결론!

객체를 멤버로 갖는 클래스

Prog

객체를 멤버로 갖는 클래스

Prog

소멸자의 호출 순서는 어떨까?

객체를 멤버로 갖는 클래스

Prog

생성자의 호출 순서와 반대!

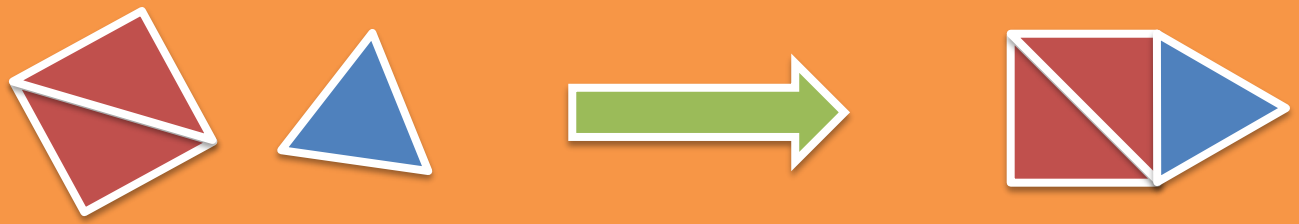
Rect 객체의 소멸자가 먼저 호출되고,

Point 객체들의 소멸자가 그 후에 호출!

객체를 멤버로 갖는 클래스

Prog

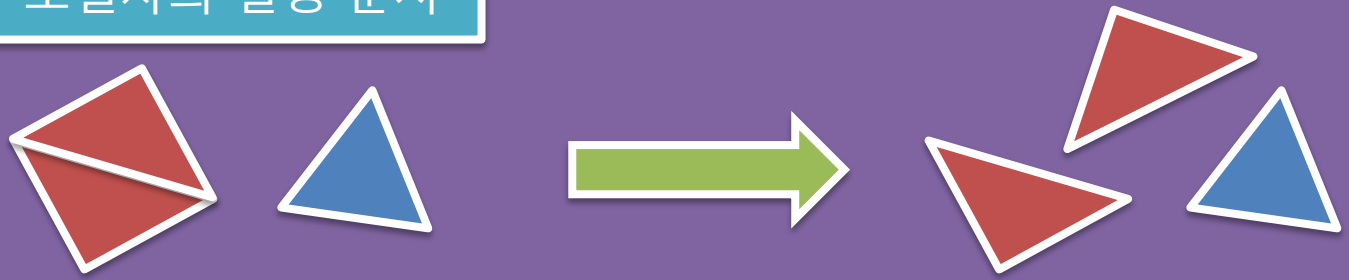
생성자의 실행 순서



Point::Point() 실행
<부품들이 먼저 실행된다>

Rect::Rect() 실행
<전체가 완성된다>

소멸자의 실행 순서



Rect::~Rect() 실행
<전체가 먼저 분해된다>

Point::~Point() 실행
<부품들이 분해된다>

5분 휴식 ^0^

문서 저장 클래스

Prog

지금부터 다룰 내용은...

기존의 클래스를 상속 받아서 새로운 클래스를 만드는 방법과

부모 클래스와 자식 클래스의 관계를 관리하는 기능!

문서 저장 클래스

Prog

자식 클래스가 부모 클래스의 모든 멤버를 물려받았다는 것만 기억한다면 나머지는 쉽게 이해할 수 있어요!

문서 저장 클래스

Prog

문서 저장 클래스

이전 장에서 객제지향 프로그래밍의 개념을 설명하면서...

예로 들었던 문서 저장 클래스와 HTML 저장 클래스...

직접 보여줄 거에요!

문서 저장 클래스

Prog

배운 지 오래되어서 기억이 안 난다고요?

문서 저장 클래스

문서 저장 클래스

Prog

구현해 봅시다!

문서 저장 클래스!

문서 저장 클래스

Prog

1. DocWriter.h

```
#ifndef DOCWRITER_H
#define DOCWRITER_H

#include <string>

class DocWriter {
public:
    DocWriter();
    DocWriter(const std::string& fileName,
        const std::string& content);
    ~DocWriter();
```

문서 저장 클래스

Prog

1. DocWriter.h

```
    // 파일 이름을 지정
    void SetFileName(const std::string& fileName);
    // 저장할 텍스트를 지정
    void SetContent(const std::string& content);
    // 파일에 텍스트를 저장
    void Write();

protected:
    std::string _fileName;
    std::string _content;
};

#endif
```

문서 저장 클래스

Prog

2. DocWriter.cpp

```
#include "DocWriter.h"
#include <fstream>

DocWriter::DocWriter() {
    // 파일 이름과 텍스트를 디폴트로 지정시켜 놓음
    _fileName = "NoName.txt";
    _content = "There is no content.";
}
```

문서 저장 클래스

2. DocWriter.cpp

```
DocWriter::DocWriter(const std::string& fileName,
    const std::string& content) {
    _fileName = fileName;
    _content = content;
}

DocWriter::~DocWriter() { }
```

문서 저장 클래스

2. DocWriter.cpp

```
// 파일 이름을 지정
void DocWriter::SetFileName(const std::string& fileName) {
    _fileName = fileName;
}

// 저장할 텍스트를 지정
void DocWriter::SetContent(const std::string& content) {
    _content = content;
}
```

문서 저장 클래스

Prog

2. DocWriter.cpp

```
// 파일에 텍스트를 저장
void DocWriter::Write() {
    // 파일을 염
    std::ofstream of(_fileName.c_str());

    // 간단한 헤더를 출력
    of << "# Content #₩n₩n";

    // 텍스트를 있는 그대로 저장
    of << _content;
}
```

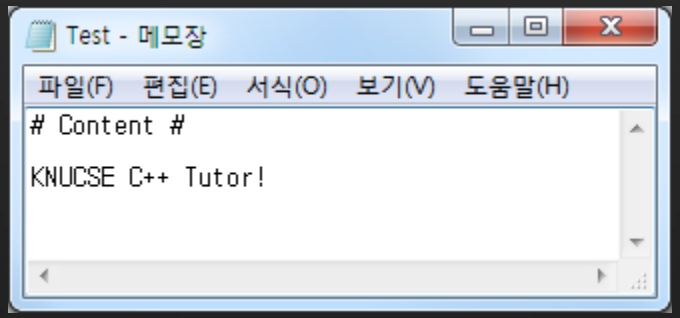
문서 저장 클래스

Prog

3. Example.cpp

```
#include "DocWriter.h"

int main() {
    DocWriter dw;
    dw.SetFileName("Test.txt");
    dw.SetContent("KNUCSE C++ Tutor!");
    dw.Write();

    return 0;
}
```

문서 저장 클래스



HTML 문서 저장 클래스

HTMLWriter 클래스는 DocWriter 클래스와 거의 비슷하지만...

크게 두 가지 차이점이 존재한다!

HTML 문서 저장 클래스

첫째, 웹 브라우저에서 열어볼 수 있는 HTML 형식으로
파일을 저장한다는 점
둘째, 웹 형식의 문서인 만큼 텍스트의 폰트를
지정할 수 있는 기능이 추가 된다는 점

HTML 문서 저장 클래스

Prog

4. HTMLWriter.h

```
#ifndef HTMLWRITER_H
#define HTMLWRITER_H

#include "DocWriter.h"

class HTMLWriter : public DocWriter {
public:
    HTMLWriter();
    ~HTMLWriter();

    // 텍스트를 파일로 저장
    void Write();
```

HTML 문서 저장 클래스

Prog

4. HTMLWriter.h

```
    // 폰트를 지정
    void SetFont(const std::string& fontName,
        int fontSize, const std::string& fontColor);

protected:
    std::string _fontName;
    int _fontSize;
    std::string _fontColor;
};

#endif
```

HTML 문서 저장 클래스

Prog

5. HTMLWriter.cpp

```
#include "HTMLWriter.h"
#include <fstream>

HTMLWriter::HTMLWriter() {
    // 디폴트 파일 이름만 바꿈
    _fileName = "NoName.html";

    // 디폴트 폰트를 지정
    _fontName = "굴림";
    _fontSize = 3;
    _fontColor = "black";
}
```

HTML 문서 저장 클래스

Prog

5. HTMLWriter.cpp

```
HTMLWriter::~HTMLWriter() { }

// 파일에 텍스트를 저장
void HTMLWriter::Write() {
    // 파일을 염
    std::ofstream of(_fileName.c_str());

    // HTML 헤더 부분을 저장
    of << "<HTML><HEAD><TITLE> ₩
        This document was generated by HTMLWriter ₩
        </TITLE></HEAD><BODY>";
    of << "<H1>Content</H1>";
```

HTML 문서 저장 클래스

Prog

5. HTMLWriter.cpp

```
    // 폰트 태그를 시작
    of << "<Font name='" << _fontName << "' size='"
       << _fontSize << "' color='" << _fontColor << "'>";

    // 텍스트를 저장
    of << _content;

    // 폰트 태그를 닫음
    of << "</FONT>";

    // HTML을 마무리
    of << "</BODY></HTML>";
}
```

HTML 문서 저장 클래스

5. HTMLWriter.cpp

```
// 폰트를 지정
void HTMLWriter::SetFont(const std::string& fontName,
    int fontSize, const std::string& fontColor) {
    _fontName = fontName;
    _fontSize = fontSize;
    _fontColor = fontColor;
}
```

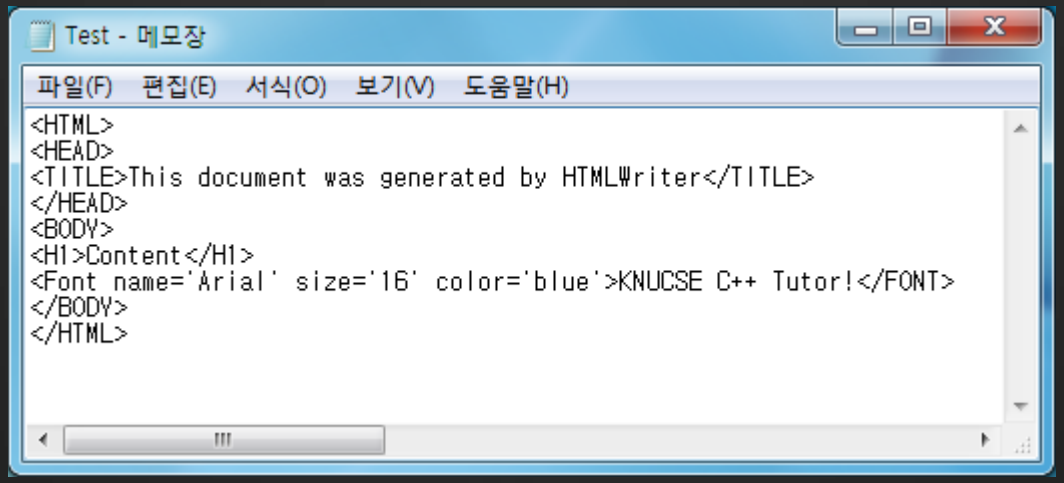
HTML 문서 저장 클래스

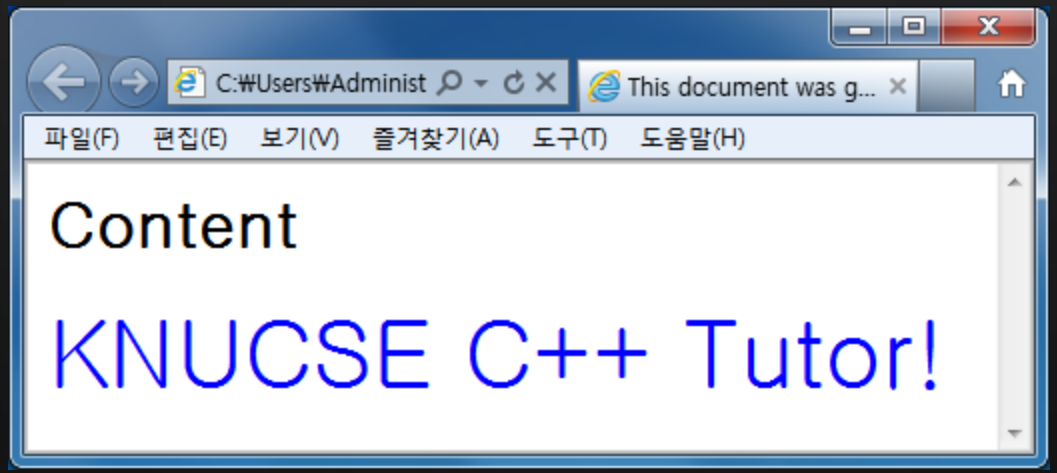
3. Example.cpp

```
#include "HTMLWriter.h"

int main() {
    HTMLWriter hw;
    hw.SetFileName("Test.html");
    hw.SetContent("KNUCSE C++ Tutor!");
    hw.SetFont("Arial", 16, "blue");
    hw.Write();

    return 0;
}
```

HTML 문서 저장 클래스

Prog





HTML 문서 저장 클래스

Prog

수많은 코드 보시느라 고생 많으셨습니다!

HTML 문서 저장 클래스

HTML 문서 저장 클래스

Prog

하지만...

HTML 문서 저장 클래스

HTML 문서 저장 클래스

Prog

클래스를 상속 받는 문법?

HTML 문서 저장 클래스

Prog

콜론을 찍어주고!

public 키워드를 적고!

클래스의 이름을 적으면 끝!

HTML 문서 저장 클래스


```
class HTMLWriter : public DocWriter {
    ...
};
```

DocWriter 클래스를
상속받는다

HTML 문서 저장 클래스

Prog

일반적으로 자식 클래스에는...

HTML 문서 저장 클래스

Prog

부모 클래스에 없는 새로운 멤버를 추가할 수도 있고

이미 존재하는 멤버 함수를 새롭게 정의할 수도 있다

HTML 문서 저장 클래스

HTML 문서 저장 클래스

Prog

HTMLWriter의 경우에는...

HTML 문서 저장 클래스

폰트 설정과 관련된 멤버 변수와 함수들을 새로 추가했다

그리고 Write() 멤버 함수를 새롭게 정의하고 있다

HTML 문서 저장 클래스

Prog

DocWriter 클래스의 객체와 HTMLWriter 클래스의 객체 메모리의 구조는 어떻게 다를까요?

HTML 문서 저장 클래스

Prog

자식 클래스의 객체는

부모 클래스가 가진 모든 멤버를 상속받죠?

HTML 문서 저장 클래스

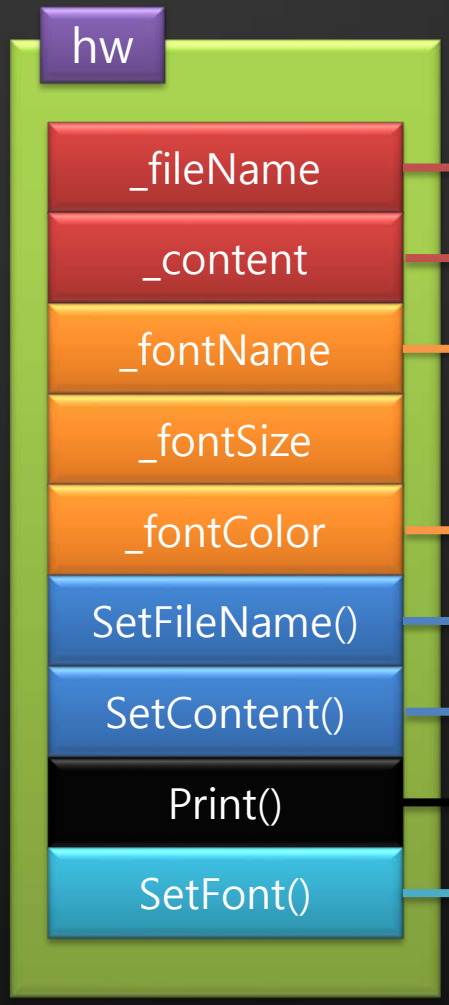
HTMLWriter 객체도

DocWriter 클래스의 모든 멤버를 소유하게 돼요!

HTML 문서 저장 클래스

DocWriter 객체

HTMLWriter 객체



DocWriter 클래스로부터 상속 받은 멤버 변수들

HTMLWriter 클래스에서 새로 추가된 멤버 변수들

DocWriter 클래스로부터 상속 받은 멤버 함수들

HTMLWriter 클래스에서 새롭게 정의한 멤버 함수

HTMLWriter 클래스에서 새로 추가된 멤버 함수

5분 휴식 ^0^

HTML 문서 저장 클래스

Prog

자식 객체를 생성할 때는...

HTML 문서 저장 클래스

자식 클래스의 생성자 뿐만 아니라

부모 클래스의 생성자도 호출된다

HTML 문서 저장 클래스

Prog

자식 객체에는 부모로부터 상속 받은 멤버들이 있는데
이 부분을 초기화하기 위해서 부모 클래스의 생성자가 필요!

HTML 문서 저장 클래스

HTML 문서 저장 클래스

Prog

앞의 예제에서는...

HTML 문서 저장 클래스

DocWriter 클래스의 어떤 생성자를 호출할지 지정하지 않았기 때문에 DocWriter 클래스의 디폴트 생성자가 호출됐다

HTML 문서 저장 클래스

DocWriter 클래스의 어떤 생성자를 호출할지
직접 지정해 봐요

HTML 문서 저장 클래스

Prog

4. HTMLWriter.h

```
HTMLWriter();
HTMLWriter(const std::string& fileName,
    const std::string& content);
~HTMLWriter();
```

HTML 문서 저장 클래스

5. HTMLWriter.cpp

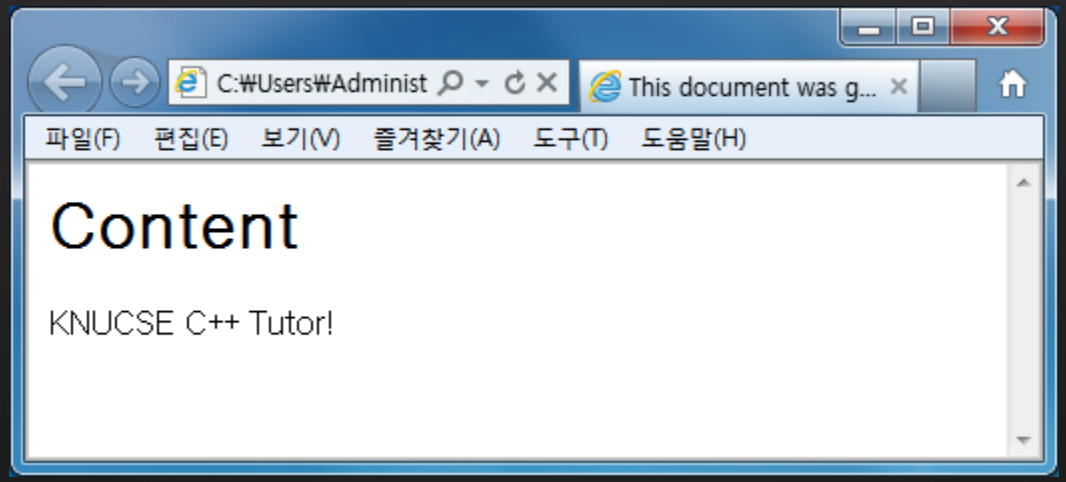
```
HTMLWriter::HTMLWriter(const std::string& fileName,
    const std::string& content)
: DocWriter(fileName, content) {
    // 디폴트 폰트를 지정
    _fontName = "굴림";
    _fontSize = 3;
    _fontColor = "black";
}
```

HTML 문서 저장 클래스

3. Example.cpp

```
#include "HTMLWriter.h"

int main() {
    HTMLWriter hw("Test.html", "KNUCSE C++ Tutor!");
    hw.Write();

    return 0;
}
```

HTML 문서 저장 클래스



HTML 문서 저장 클래스

Prog

멤버인 객체를 초기화할 때는 객체의 이름을 사용했지만

부모 클래스를 초기화할 때는 클래스의 이름을 사용!

HTML 문서 저장 클래스

Prog

```
HTMLWriter::HTMLWriter(const std::string& fileName,
    const std::string& content)
: DocWriter(fileName, content) {
    ...
}
```

부모 클래스 이름을 사용해서
생성자를 호출한다

HTML 문서 저장 클래스

Prog

여기서 알 수 있는 사실 하나!

HTML 문서 저장 클래스

초기화 리스트에서 부모 클래스의 생성자를 호출하기 때문에...

HTML 문서 저장 클래스

Prog

부모 클래스의 생성자가

자식 클래스의 생성자보다 먼저 호출된다!

HTML 문서 저장 클래스

Prog

소멸자의 호출 순서는 반대!

HTML 문서 저장 클래스

Prog

자식 클래스에 해당하는 부분이 먼저 정리된 후에,

부모 클래스에 해하는 부분이 정리된다!

HTML 문서 저장 클래스

HTML 문서 저장 클래스

Prog

간단한 예제로 확인해 봐요

HTML 문서 저장 클래스

Prog

InheritanceCallTest.cpp

```
class Base {
public:
    Base();
    ~Base();
};

Base::Base() {
    std::cout << "Base::Base() 호출!₩n";
}
Base::~Base() {
    std::cout << "Base::~Base() 호출!₩n";
}
```

HTML 문서 저장 클래스

Prog

InheritanceCallTest.cpp

```
class Derived : public Base {
public:
    Derived();
    ~Derived();
};

Derived::Derived() {
    std::cout << "Derived::Derived() 호출!₩n";
}
Derived::~Derived() {
    std::cout << "Derived::~Derived() 호출!₩n";
}
```

HTML 문서 저장 클래스

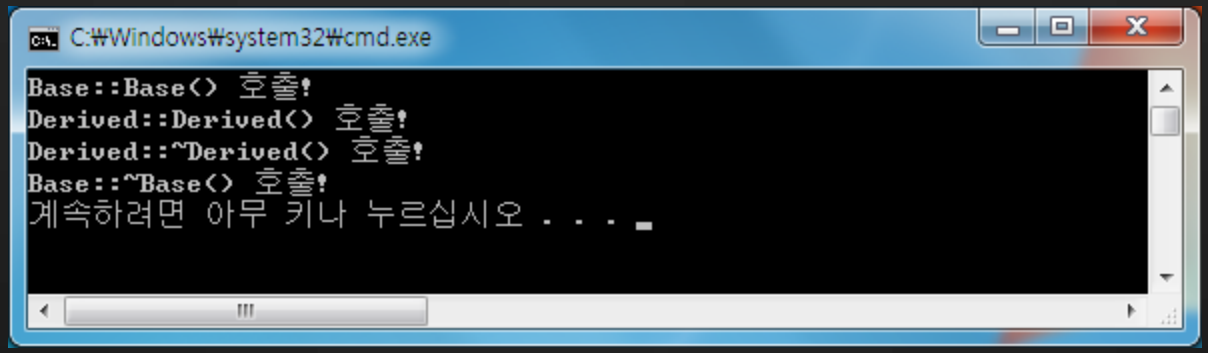
Prog

InheritanceCallTest.cpp

```
int main() {
    Derived d;

    return 0;
}
```

HTML 문서 저장 클래스

Prog



HTML 문서 저장 클래스

HTML 문서 저장 클래스

Prog

정리해 봐요!

HTML 문서 저장 클래스

Prog

자식 객체의 생성 시

부모 클래스의 생성자 → 자식 클래스의 생성자

HTML 문서 저장 클래스

자식 객체의 소멸 시

자식 클래스의 소멸자 → 부모 클래스의 소멸자

클래스간의 형변환

Prog

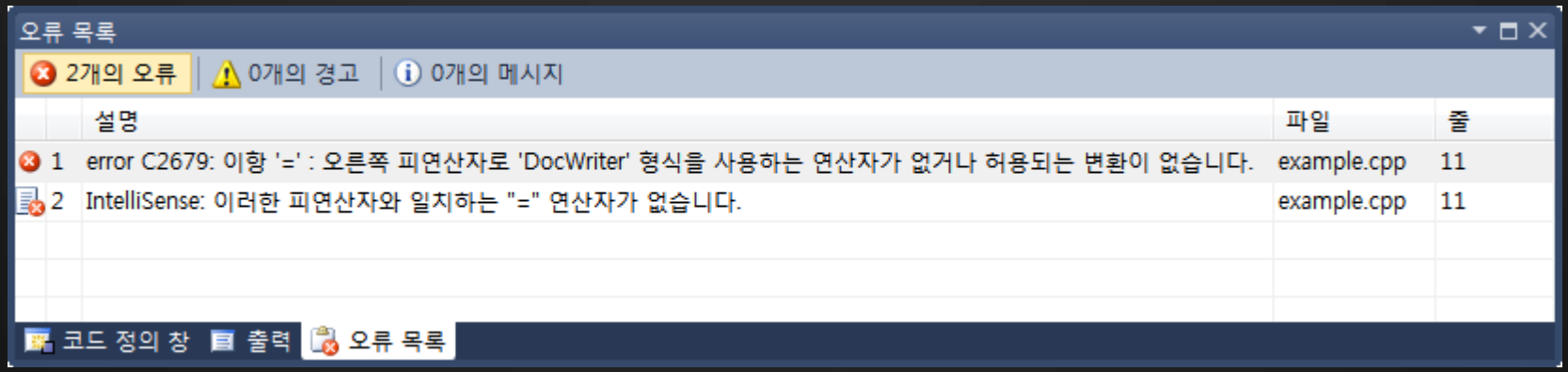
우선 부모 객체에서 자식 객체로의 대입을 살펴봐요

클래스간의 형변환

3. Example.cpp

```
int main() {
    // 자식 클래스의 객체 생성
    HTMLWriter hw("Test.html", "HTMLWriter Content");
    // 부모 클래스의 객체 생성
    DocWriter dw;

    // 부모 클래스의 객체를 자식 클래스의 객체로 대입
    dw = hw;
    // 파일 저장
    hw.Write();

    return 0;
}
```

클래스간의 형변환



클래스간의 형변환

왜 오류가 날까요?

클래스간의 형변환

Prog

힌트

클래스간의 형변환

객체의 대입이란 사실 모든 멤버 변수들의 1 : 1 대입을 의미

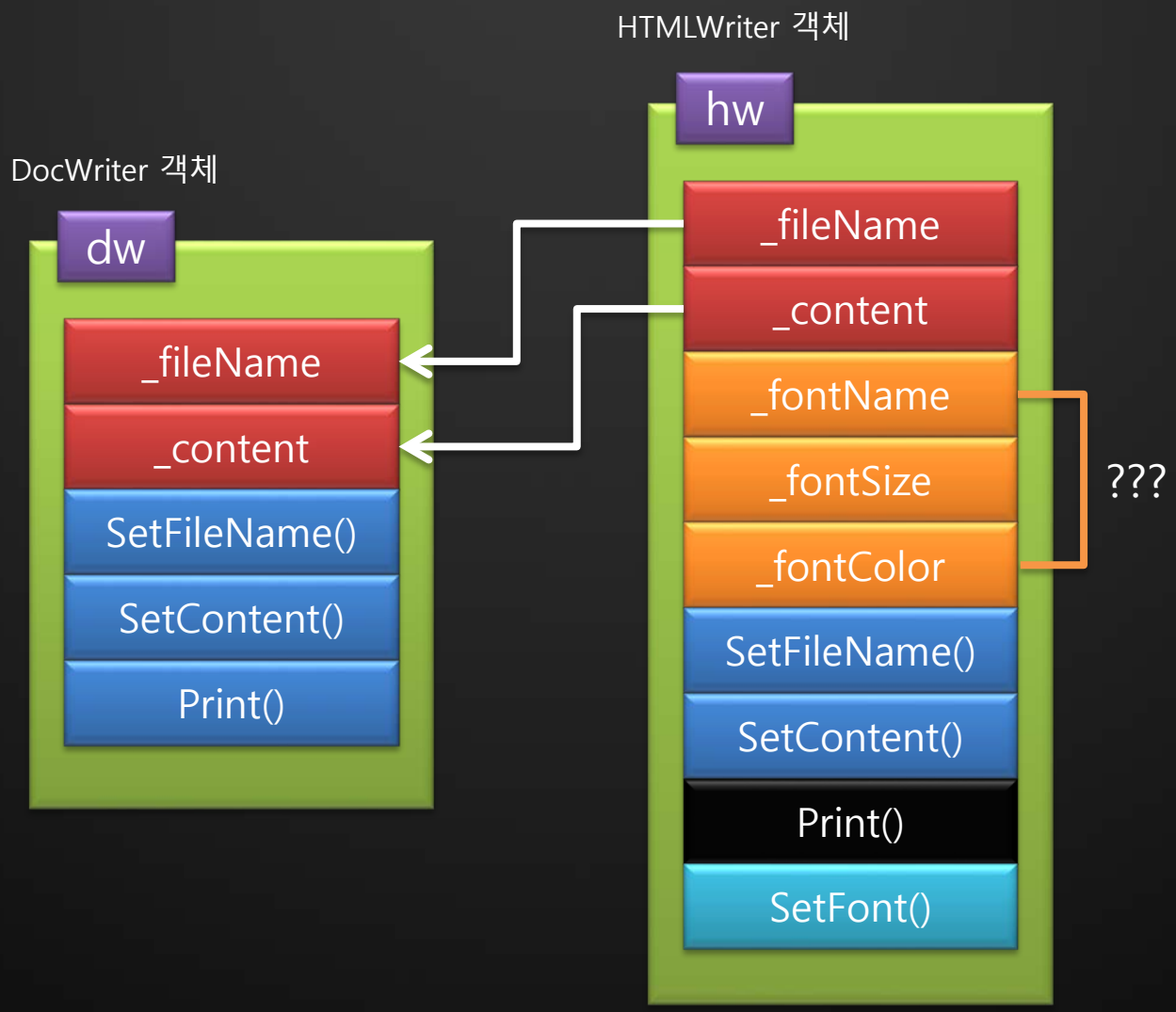
클래스간의 형변환

Prog

그런데 자식 객체에는 부모 객체에 없는 멤버가 있다

(_fontName, _fontSize, _fontColor와 같은 멤버들)

클래스간의 형변환



클래스간의 형변환

Prog

그러면 반대의 경우는 어떨까요?

(자식 객체를 부모 객체로 대입하는 경우)

클래스간의 형변환

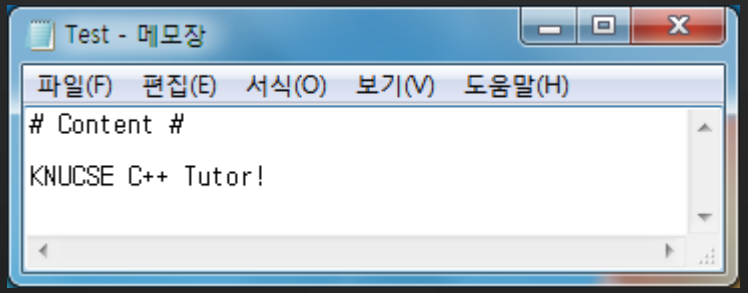
Prog

3. Example.cpp

```
int main() {
    // 자식 클래스의 객체 생성
    HTMLWriter hw("Test.html", "HTMLWriter Content");
    // 부모 클래스의 객체 생성
    DocWriter dw;

    // 부모 클래스의 객체를 자식 클래스의 객체로 대입
    hw = dw;
    // 파일 저장
    hw.Write();

    return 0;
}
```

클래스간의 형변환

Prog



클래스간의 형변환

Prog

부모 객체와 자식 객체에 공통적으로 있는 멤버들이 1 : 1로 대입!

클래스간의 형변환

Prog

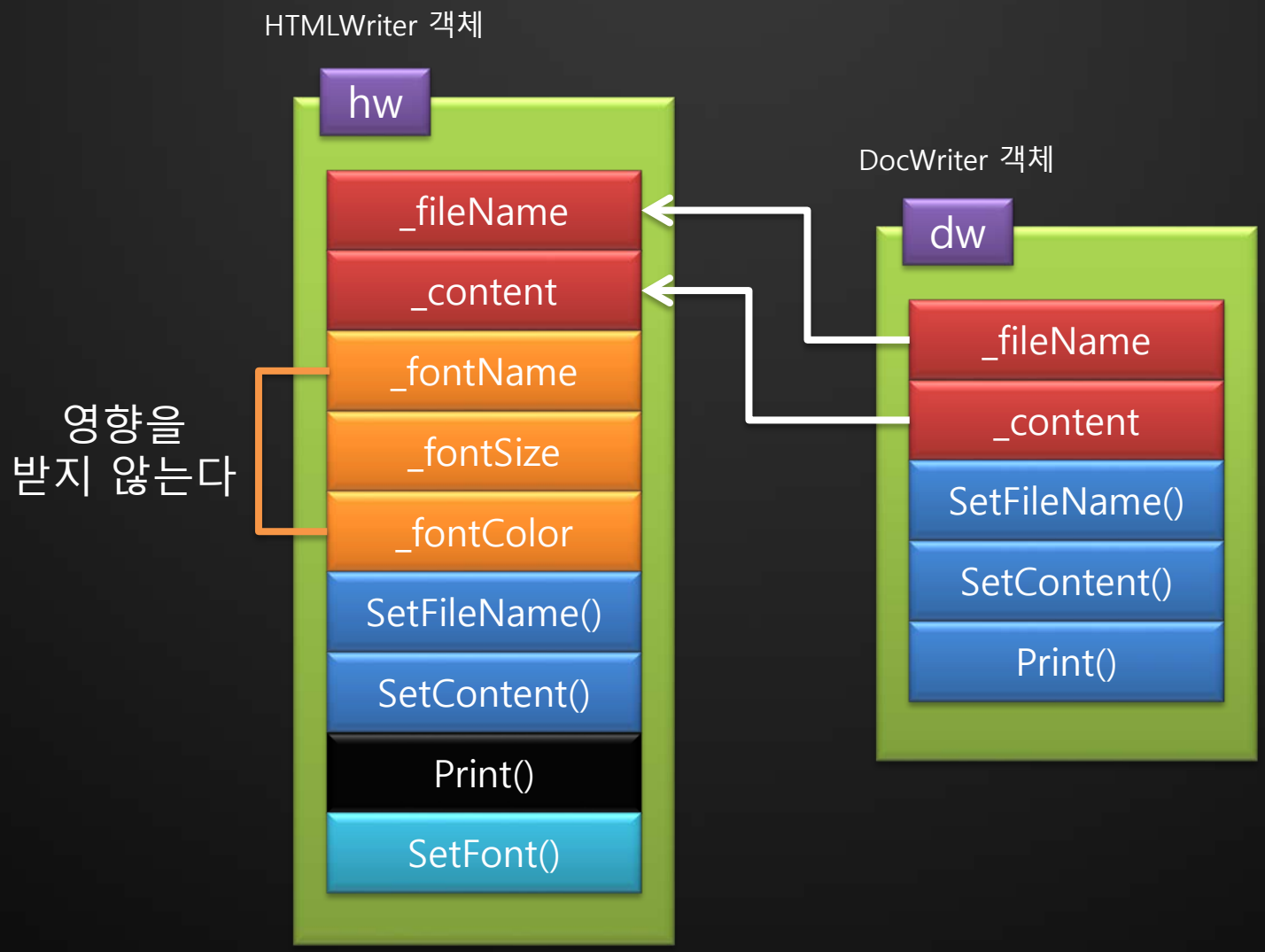
자식 객체에만 있는 멤버들은?

클래스간의 형변환

Prog

아무런 영향을 미치지 못한다!

클래스간의 형변환



5분 휴식 ^0^

클래스간의 형변환

Prog

이번에는 객체의 포인터간의 형변환과

레퍼런스간의 형변환을 살펴봐요!

클래스간의 형변환

Prog

자식 클래스의 포인터를 사용해서 부모 객체를 가리키는 경우

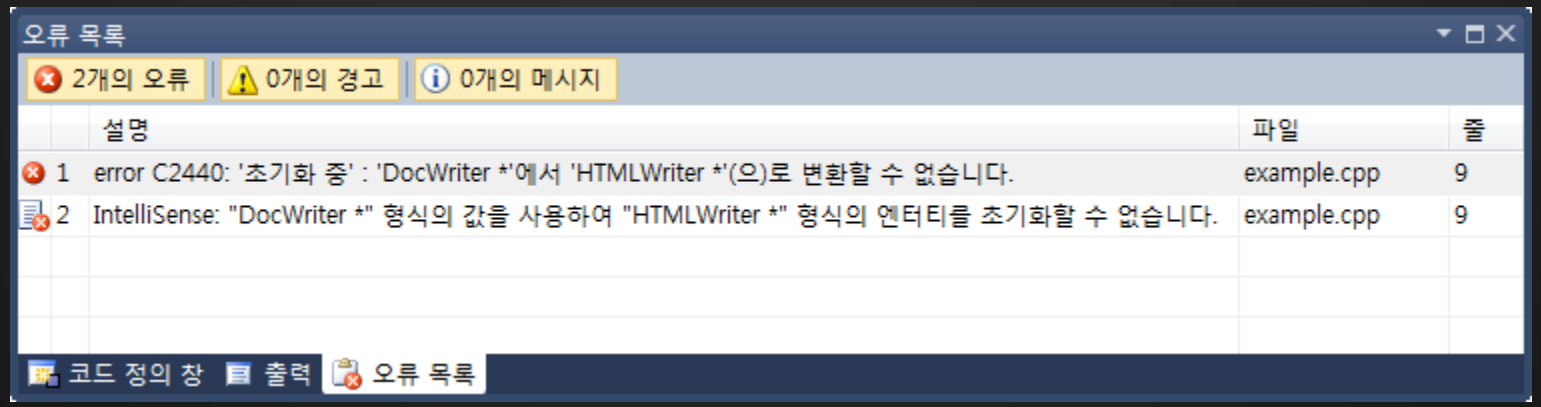
클래스간의 형변환

Prog

3. Example.cpp

```
int main() {
    // 부모 객체 생성
    DocWriter dw("Test.txt", "DocWriter Content");

    // 자식 클래스의 포인터로 부모 객체를 가리킴
    HTMLWriter* phw = &dw;

    // 파일에 저장
    phw->Write();

    return 0;
}
```

클래스간의 형변환



클래스간의 형변환

Prog

자식 클래스의 포인터를 사용해서 부모 객체를 가리키는 경우

암시적인 형변환을 허용하지 않는다

클래스간의 형변환

컴퓨터가 형변환의 안전성을 보장할 수 없기 때문!

클래스간의 형변환

Prog

컴퓨터 입장에서는...

클래스간의 형변환

Prog

클래스간의 형변환

Prog

하지만...

형변환을 허용한다고 가정해보죠!

클래스간의 형변환

클래스간의 형변환

Prog

포인터가 아니라 레퍼런스를 사용하면 어떨까요?

클래스간의 형변환

3. Example.cpp

```
int main() {
    // 부모 객체 생성
    DocWriter dw("Test.txt", "DocWriter Content");

    // 자식 클래스의 포인터로 부모 객체를 가리킴
    HTMLWriter& rhw = dw;

    // 파일에 저장
    rhw.Write();

    return 0;
}
```

클래스간의 형변환

오류 목록

2개의 오류 | 0개의 경고 | 0개의 메시지

	설명	파일	줄
1	error C2440: '초기화 중' : 'DocWriter'에서 'HTMLWriter &'(으)로 변환할 수 없습니다.	example.cpp	9
2	IntelliSense: "HTMLWriter &" 형식(const 한정 형식 아님)의 참조를 "DocWriter" 형식의 값으로 초기화할 수 없습니다.	example.cpp	9

코드 정의 창 | 출력 | 오류 목록

클래스간의 형변환

Prog

클래스간의 형변환

Prog

하지만 반대의 경우는 허용된다!

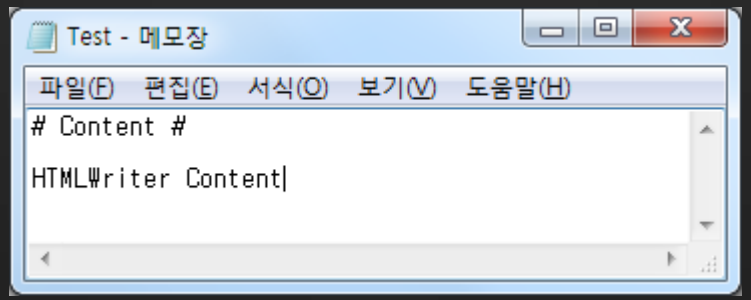
(부모 클래스의 포인터로 자식 객체를 가리키는 경우)

클래스간의 형변환

3. Example.cpp

```
int main() {
    // HTMLWriter 객체 생성
    HTMLWriter hw("Test.html", "HTMLWriter Content");

    // DocWriter 클래스의 포인터로 객체를 가리킴
    DocWriter* pdw = &hw;

    // 파일에 저장
    pdw->Write();

    return 0;
}
```

클래스간의 형변환



클래스간의 형변환

Prog

부모 클래스의 포인터로 자식 객체를 가리키는 경우에는

문제가 생길 여지가 없다

(부모 클래스에 있는 모든 멤버는 자식 객체에도 있기 때문)

클래스간의 형변환

클래스간의 형변환

Prog

하지만 이상해요

저장한 문서의 타입이 HTML 포맷이 아닌 그냥 텍스트 포맷!

클래스간의 형변환

Prog

그것은 DocWriter::Write() 멤버 함수가 호출됐기 때문!

클래스간의 형변환

Prog

사실 DocWriter::Write()를 호출해야 한다는 주장과

HTMLWriter()::Write()를 호출해야 한다는 주장 모두 가능하다

클래스간의 형변환

Prog

pdw의 타입이
DocWriter*이기 때문에
DocWriter::Write() 함수가
호출되어야 해!

pdw가 가리키는 객체는
실제로 HTMLWriter이기 때문에
HTMLWriter::Write() 함수가
호출되어야 해!

클래스간의 형변환

Prog

하지만 사실은...

클래스간의 형변환

Prog

야! 개 짖는 소리 좀 안 나게 해라!!

클래스간의 형변환

Prog

이게 아니지...

클래스간의 형변환

Prog

"실제 객체가
무엇이던 간에 상관 없이
포인터의 타입을 기준으로
호출될 함수가 결정된다."

클래스간의 형변환

Prog

방금 예제에서는
DocWriter::Write()가 호출됐잖아

클래스간의 형변환

Prog

하지만...

클래스간의 형변환

Prog

훼이크다 이 병신들아!

클래스간의 형변환

Prog

다음주에 가상 함수를 배우면
HTMLWriter::Write()가 호출되는
진풍경을 볼 수 있을꺼야

클래스간의 형변환

Prog

그건 다음주에 알아보고...
레퍼런스의 경우를 살펴보고
형변환 얘기는 여기서 끝내자구!

클래스간의 형변환

Prog

3. Example.cpp

```
int main() {
    // HTMLWriter 객체 생성
    HTMLWriter hw("Test.html", "HTMLWriter Content");

    // DocWriter 클래스의 레퍼런스로 객체를 참조함
    DocWriter& dw = hw;

    // 파일에 저장
    dw.Write();

    return 0;
}
```

접근 제어

Prog

접근 제어 키워드가 의미하는 바를 다시 정리해 보면...

접근 제어

Prog

public : 모든 곳으로부터의 접근을 허용한다

protected : 자식 클래스의 멤버 함수로부터의 접근만 허용한다

private : 자신의 멤버 함수 외에는 접근할 수 없다

접근 제어

Prog

예제를 보면서 구체적으로 확인해 봐요!

접근 제어

AccessControl.cpp

```
class Base {
private:
    int priv;

protected:
    int prot;

public:
    int pub;
};
```

접근 제어

Prog

AccessControl.cpp

```
class Dervied : public Base {
public:
    void AccessBases() {
        // 부모의 멤버에 접근을 시도
        int n;
        n = priv;
        n = prot;
        n = pub;
    }
};
```

접근 제어

AccessControl.cpp

```
int main() {
    Base base;

    // 객체의 멤버에 접근을 시도
    int n;
    n = base.priv;
    n = base.prot;
    n = base.pub;

    return 0;
}
```

접근 제어

Prog

오류 목록

7개의 오류 | 0개의 경고 | 0개의 메시지

	설명	파일	줄
1	error C2248: 'Base::priv' : private 멤버('Base' 클래스에서 선언)에 액세스할 수 없습니다.	test.cpp	15
2	error C2248: 'Base::priv' : private 멤버('Base' 클래스에서 선언)에 액세스할 수 없습니다.	test.cpp	26
3	error C2248: 'Base::prot' : protected 멤버('Base' 클래스에서 선언)에 액세스할 수 없습니다.	test.cpp	27
4	IntelliSense: PCH 경고: 알맞은 헤더 중지 위치를 찾을 수 없습니다. intellisense PCH 파일이 생성되지 않았습니다.	test.cpp	1
5	IntelliSense: 멤버 "Base::priv" (선언됨 줄 3)에 액세스할 수 없습니다.	test.cpp	15
6	IntelliSense: 멤버 "Base::priv" (선언됨 줄 3)에 액세스할 수 없습니다.	test.cpp	26
7	IntelliSense: 멤버 "Base::prot" (선언됨 줄 5)에 액세스할 수 없습니다.	test.cpp	27

접근 제어

Prog

각 상황별로 정리해보죠!

접근 제어

Prog

	자신의 멤버 함수에서 접근	자식 클래스의 멤버 함수에서 접근	외부에서 접근
private 멤버	O	X	X
protected 멤버	O	O	X
public 멤버	O	O	O

접근 제어

Prog

그런데 어떻게 활용하죠?

접근 제어

접근 제어

Prog

좌절하지 마세요!

가이드라인을 제시해 드릴게요!

접근 제어

Prog

외부로부터 숨겨야 하는 멤버는 protected로 지정한다

그 밖의 경우는 public으로 지정한다

반드시 자식 클래스에 숨기고 싶다면 private로 지정한다

접근 제어

Prog

9주차 레포트

- 업 캐스트와 다운 캐스트에 대하여...
  - 자식 클래스의 포인터가 부모 클래스의 포인터로 형변환하는 것을 업 캐스트라고 한다
  - 부모 클래스의 포인터에서 자식 클래스의 포인터로 형변환하는 것을 다운 캐스트라고 한다
- 다중 상속에 대하여...
  - 다중 상속을 사용하는 클래스 예제를 하나 만들고, 문제점을 파악!
  - 다중 상속(Multiple Inheritance)은 두 개 이상의 부모 클래스를 동시에 상속하는 경우를 말한다
- 포함과 상속을 구분하는 방법에 대하여...
  - 예를 들어서 설명! (Has-a : 자동차와 타이어, Is-a : 사과와 과일)
  - 포함은 Has-a 관계 : A가 B를 가지고 있다면(Has) 포함을 사용
  - 상속은 Is-a 관계 : A가 B라면(Is) 상속을 사용

이에 대한 자신의 생각을 A4 종이 1장 이내로 작성하여 다음주에 제출

9주차 숙제

- 버그 수정
  - Rect 클래스에는 문제점이 존재
  - 예를 들어 _topLeft의 값이 (10, 10)이고 _bottomRight의 값이 (0, 0)이라면, GetWidth() 함수와 GetHeight() 함수는 각각 -10을 반환
  - 이런 경우에도 양수의 값을 반환하게 수정
- 상속을 고려한 생성자 구현
  - Shape.cpp 파일을 참고하여 Triangle 클래스의 생성자를 구현
  - 반드시 초기화 리스트를 사용할 것!
  - Shape.cpp 파일은 C++ 튜터 클럽에 업로드 예정

10주차 수업 안내

- 다형성과 가상 함수
  - 가상 함수를 사용한 다형성의 구현
    - 언제 가상 함수가 필요할까
    - 문서 저장 클래스에 다형성 적용하기
  - 오버라이딩
    - 순수 가상 함수
    - 다양한 종류의 멤버 함수
    - 오버로딩과 오버라이딩

THANK YOU!