

9주차 수업 정리노트

2023. 10. 30 (월)
16조 이수영, 원준서

목차

- **프렌드 함수**
- 예제 7-1/2/3
- **연산자 중복**
- 연산자를 멤버 함수로 표현
- 연산자를 프렌드 함수로 표현
- 이항 연산자 중복
- 단항 연산자 중복
- 단항 연산자-전위 연산자
- 단항 연산자-후위 연산자

프렌드 함수

- 원준서: 클래스의 멤버함수가 아닌 외부 함수로, 전역 함수나 다른 클래스의 멤버 함수입니다.
- friend 키워드로 클래스 내에 선언된 함수로, 클래스의 모든 멤버를 접근할 수 있는 권한을 부여합니다. friend 함수라고도 불립니다. 프렌드 선언의 필요성으로는 클래스의 멤버로 선언하기에는 무리가 있고, 모든 멤버를 자유롭게 접근할 수 있는 일부 외부 함수를 작성할 때 필요합니다.
- 프렌드 함수가 되는 3가지로는 전역 함수, 다른 클래스의 멤버 함수, 다른 클래스 전체입니다.
- 클래스의 멤버로 선언하기 힘들지만, 클래스의 모든 멤버를 자유롭게 접근하는 일부 외부 함수를 작성할 경우 프렌드 함수를 사용하면 된다는 것을 새롭게 알게 되었습니다.

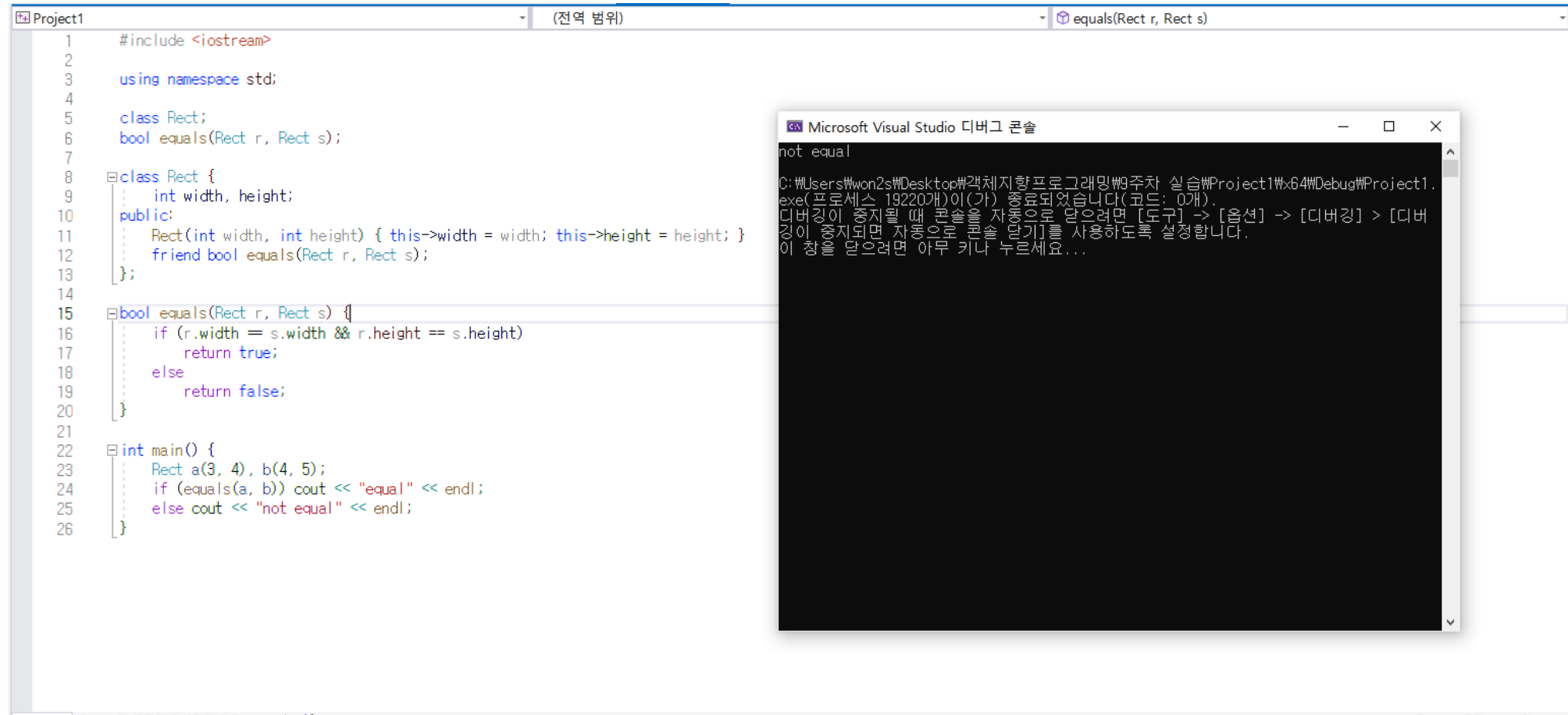
프렌드 함수

- 이수영: 프렌드 함수는 클래스 외부에 있는데, 클래스 내 멤버에 접근할 수 있는 권한을 가진 함수입니다. 프렌드 함수는 있어도 프렌드 변수라는 말은 없습니다.
- 같은 이름의 equals() 함수도 3가지로 프렌드 선언될 수 있는데, friend bool equals(...)는 전역 함수 equals가 클래스 내 멤버에 접근할 수 있는 경우입니다.

friend bool RectManager::equals(...)는 클래스 외부의 RectManager 클래스의 equals()라는 함수가 클래스 내 멤버에 접근할 수 있는 경우입니다.

마지막으로 friend RectManager;와 같이 클래스명(RectManager) 자체가 프렌드 함수로 선언되기도 합니다. 앞의 2가지는 유추됐는데 마지막처럼 클래스 자체가 프렌드 함수가 될 수 있다는 점이 새로워서 기억합니다.

예제 7-1: 원준서



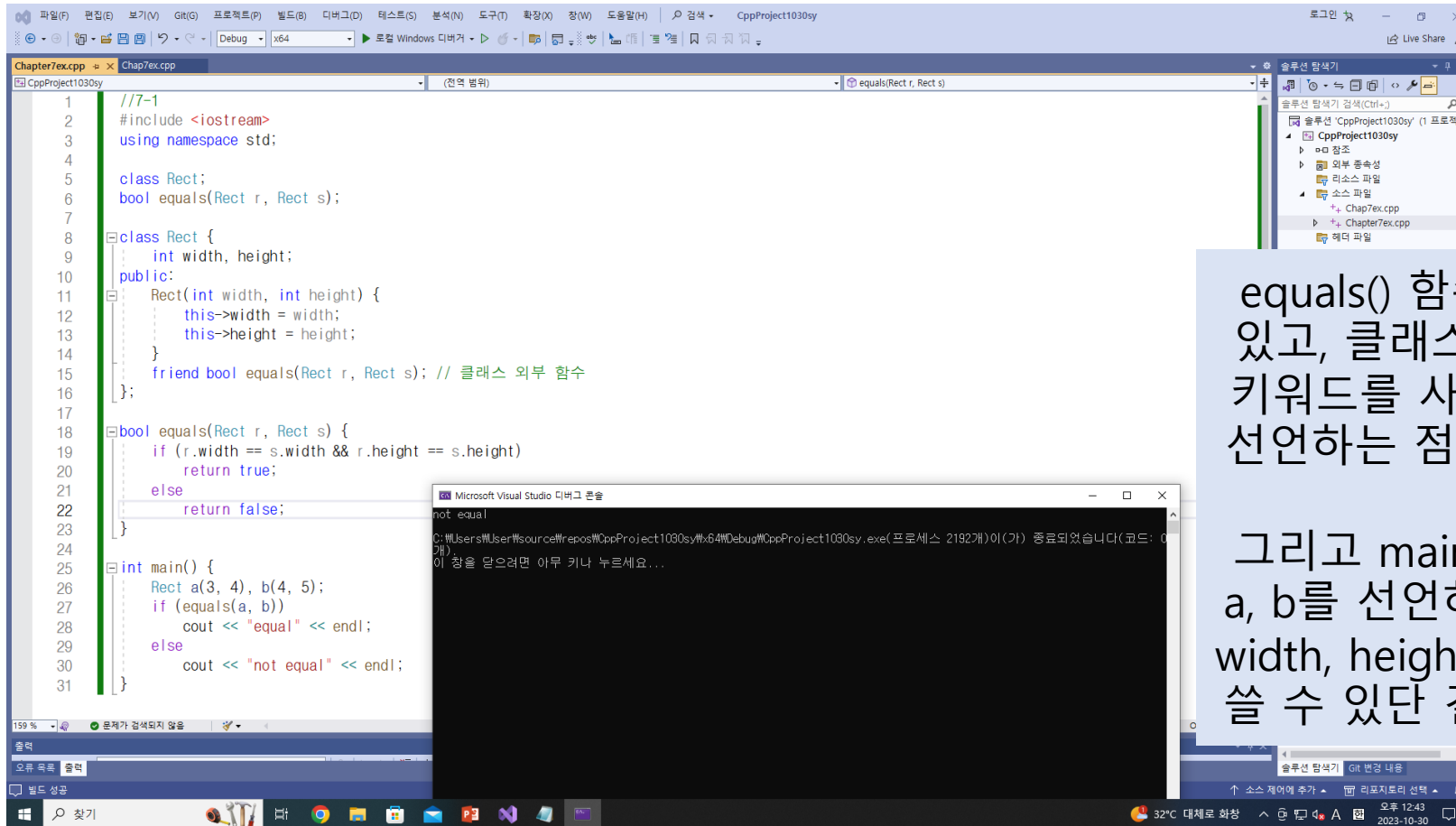
```
Project1 (전역 범위) equals(Rect r, Rect s)
```

```
1  #include <iostream>
2
3  using namespace std;
4
5  class Rect;
6  bool equals(Rect r, Rect s);
7
8  class Rect {
9  public:
10     int width, height;
11     Rect(int width, int height) { this->width = width; this->height = height; }
12     friend bool equals(Rect r, Rect s);
13 };
14
15 bool equals(Rect r, Rect s) {
16     if (r.width == s.width && r.height == s.height)
17         return true;
18     else
19         return false;
20 }
21
22 int main() {
23     Rect a(3, 4), b(4, 5);
24     if (equals(a, b)) cout << "equal" << endl;
25     else cout << "not equal" << endl;
26 }
```

Microsoft Visual Studio 디버그 콘솔

```
not equal
C:\Users\won2s\Desktop\객체지향프로그래밍\주차 실습\Project1\Debug\Project1.exe (프로세스 19220개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```

예제 7-1: 이수영



```
//7-1
#include <iostream>
using namespace std;

class Rect;
bool equals(Rect r, Rect s);

class Rect {
    int width, height;
public:
    Rect(int width, int height) {
        this->width = width;
        this->height = height;
    }
    friend bool equals(Rect r, Rect s); // 클래스 외부 함수
};

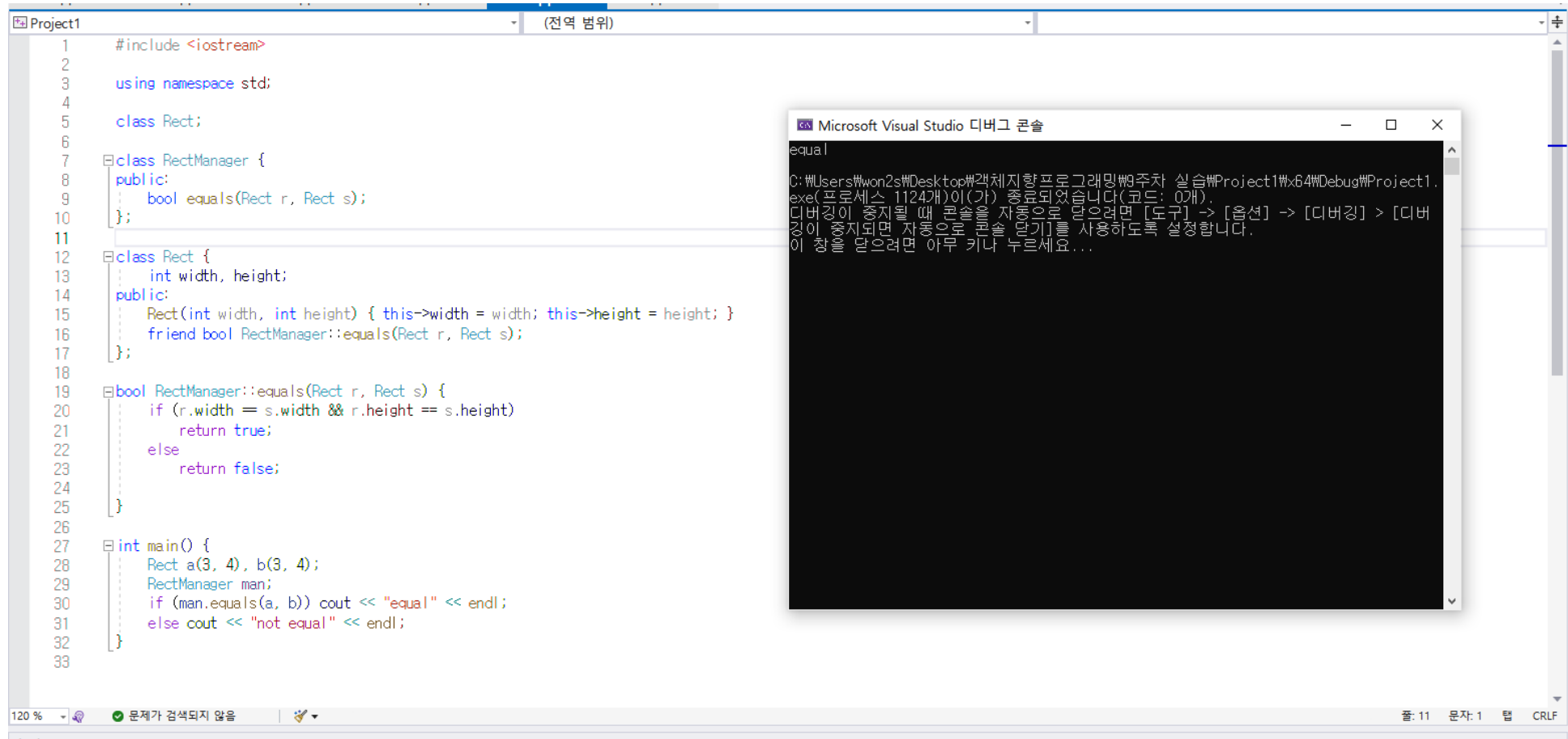
bool equals(Rect r, Rect s) {
    if (r.width == s.width && r.height == s.height)
        return true;
    else
        return false;
}

int main() {
    Rect a(3, 4), b(4, 5);
    if (equals(a, b))
        cout << "equal" << endl;
    else
        cout << "not equal" << endl;
}
```

equals() 함수가 클래스 외부에 있고, 클래스 선언부에서 friend 키워드를 사용해 프렌드 함수를 선언하는 점을 새로 배웠습니다.

그리고 main()에서 클래스 객체 a, b를 선언하고, 그 객체에 맞는 width, height라는 private 변수를 쓸 수 있단 걸 알 수 있었습니다.

예제 7-2: 원준서

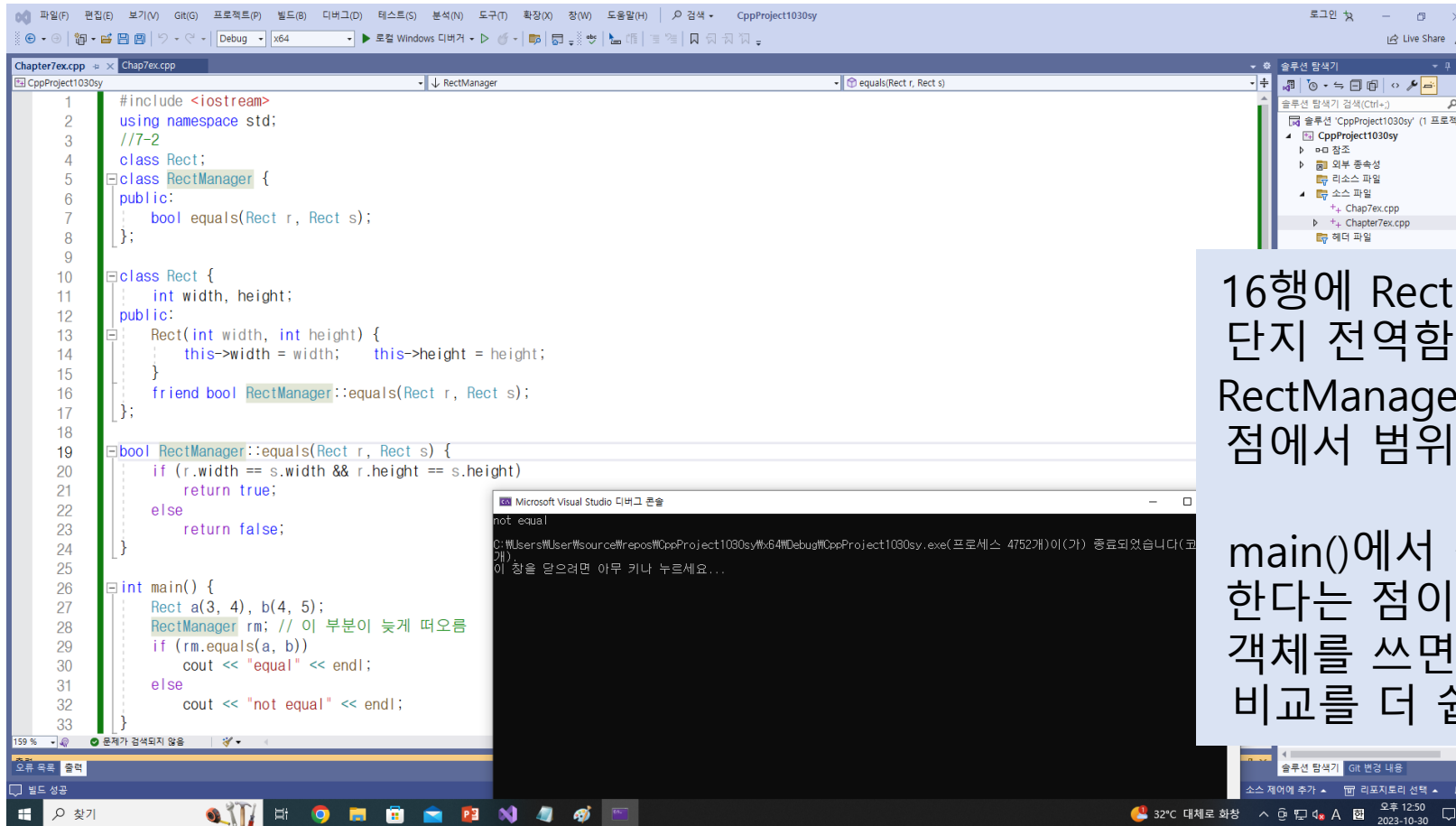


The image shows a screenshot of the Microsoft Visual Studio IDE. The main window displays a C++ source file for a project named "Project1". The code defines a `Rect` class and a `RectManager` class. The `Rect` class has `width` and `height` attributes and a constructor. The `RectManager` class has a static `equals` method that compares two `Rect` objects. The `main` function creates two `Rect` objects, `a` and `b`, and uses the `RectManager::equals` method to check if they are equal. The output of the program is "equal".

```
1  #include <iostream>
2
3  using namespace std;
4
5  class Rect;
6
7  class RectManager {
8  public:
9      bool equals(Rect r, Rect s);
10 };
11
12 class Rect {
13     int width, height;
14 public:
15     Rect(int width, int height) { this->width = width; this->height = height; }
16     friend bool RectManager::equals(Rect r, Rect s);
17 };
18
19 bool RectManager::equals(Rect r, Rect s) {
20     if (r.width == s.width && r.height == s.height)
21         return true;
22     else
23         return false;
24 }
25
26
27 int main() {
28     Rect a(3, 4), b(3, 4);
29     RectManager man;
30     if (man.equals(a, b)) cout << "equal" << endl;
31     else cout << "not equal" << endl;
32 }
33
```

The debug console window, titled "Microsoft Visual Studio 디버그 콘솔", shows the output of the program: "equal". Below the output, there is a message in Korean: "C:\Users\won2s\Desktop\객체지향프로그래밍\주차 실습\Project1\Debug\Project1.exe (프로세스 1124개)이(가) 종료되었습니다(코드: 0개). 디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다. 이 창을 닫으려면 아무 키나 누르세요..."

예제 7-2: 이수영



```
1 #include <iostream>
2 using namespace std;
3 //7-2
4 class Rect;
5 class RectManager {
6 public:
7     bool equals(Rect r, Rect s);
8 };
9
10 class Rect {
11     int width, height;
12 public:
13     Rect(int width, int height) {
14         this->width = width; this->height = height;
15     }
16     friend bool RectManager::equals(Rect r, Rect s);
17 };
18
19 bool RectManager::equals(Rect r, Rect s) {
20     if (r.width == s.width && r.height == s.height)
21         return true;
22     else
23         return false;
24 }
25
26 int main() {
27     Rect a(3, 4), b(4, 5);
28     RectManager rm; // 이 부분이 늦게 떠오름
29     if (rm.equals(a, b))
30         cout << "equal" << endl;
31     else
32         cout << "not equal" << endl;
33 }
```

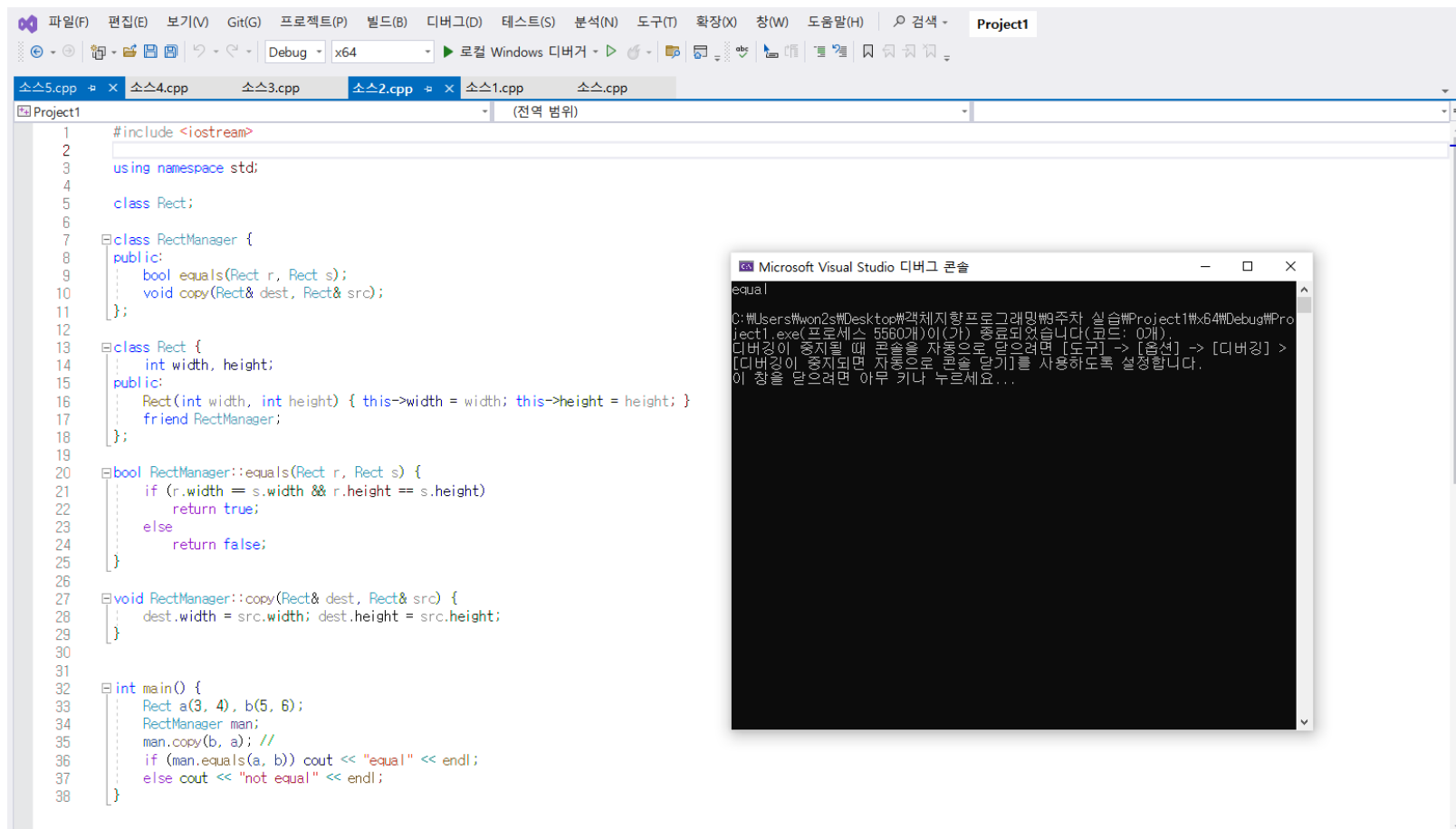
Microsoft Visual Studio 디버그 콘솔

```
not equal
C:\Users\User\source\repos\CppProject1030sy\Debug\CppProject1030sy.exe (프로세스 4752개)이(가) 종료되었습니다(코
이 창을 닫으려면 아무 키나 누르세요...
```

16행에 RectManager::가 추가됨.
단지 전역함수 equals(...)가 아닌
RectManager 클래스의 함수라는
점에서 범위지정 연산자가 쓰임.

main()에서 객체 rm을 선언해야
한다는 점이 늦게 생각났습니다.
객체를 쓰면, rm.equals()과 같이
비교를 더 쉽게 할 수 있습니다.

예제 7-3: 원준서



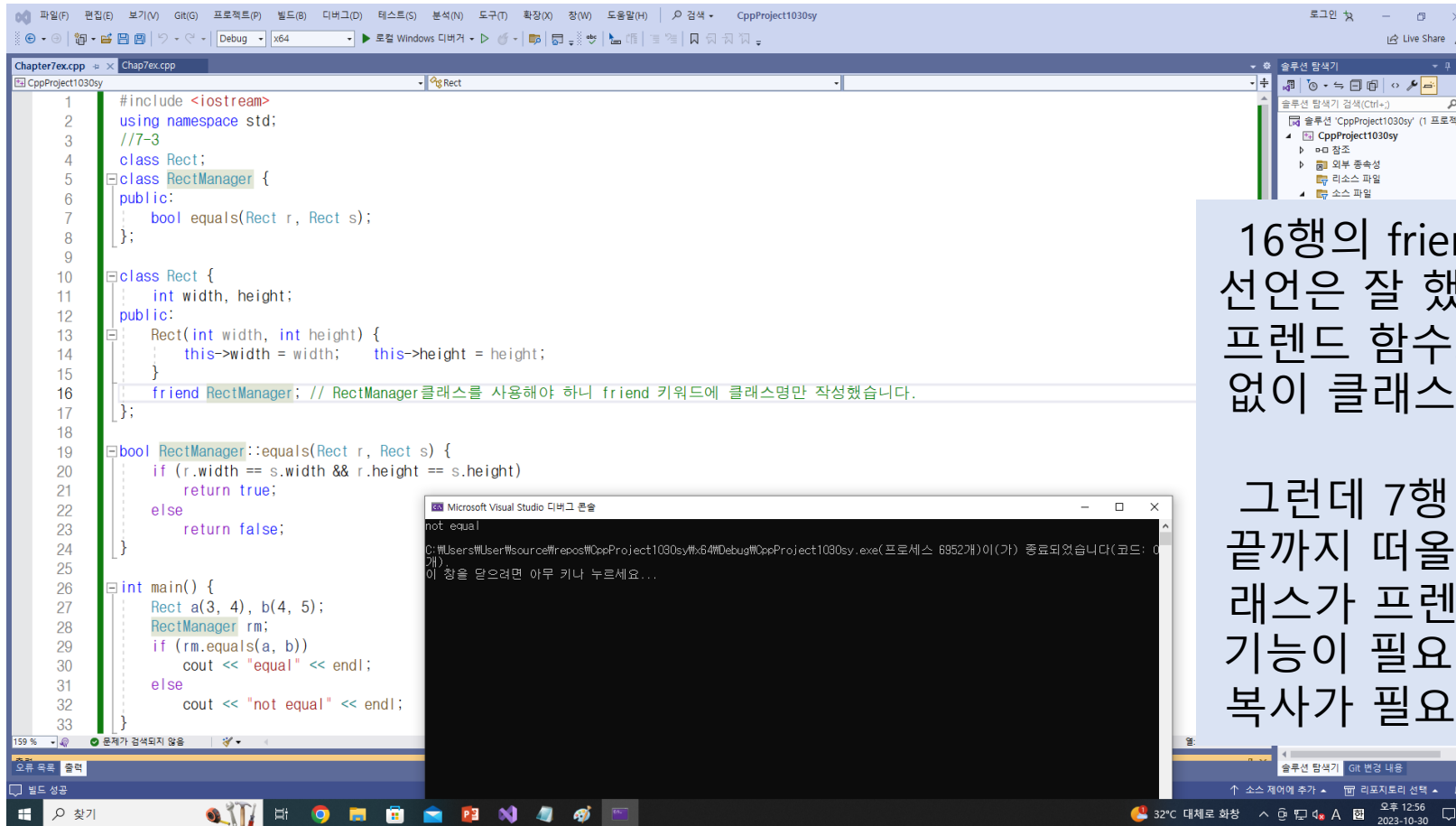
The image shows a screenshot of the Microsoft Visual Studio IDE. The main window displays a C++ source file named '소스2.cpp' (Source2.cpp) with the following code:

```
1  #include <iostream>
2
3  using namespace std;
4
5  class Rect;
6
7  class RectManager {
8  public:
9      bool equals(Rect r, Rect s);
10     void copy(Rect& dest, Rect& src);
11 };
12
13 class Rect {
14     int width, height;
15 public:
16     Rect(int width, int height) { this->width = width; this->height = height; }
17     friend RectManager;
18 };
19
20 bool RectManager::equals(Rect r, Rect s) {
21     if (r.width == s.width && r.height == s.height)
22         return true;
23     else
24         return false;
25 }
26
27 void RectManager::copy(Rect& dest, Rect& src) {
28     dest.width = src.width; dest.height = src.height;
29 }
30
31
32 int main() {
33     Rect a(3, 4), b(5, 6);
34     RectManager man;
35     man.copy(b, a); //
36     if (man.equals(a, b)) cout << "equal" << endl;
37     else cout << "not equal" << endl;
38 }
```

Overlaid on the bottom right is a 'Microsoft Visual Studio 디버거 콘솔' (Debugger Console) window. It displays the following text:

```
equal
C:\Users\won2s\Desktop\객체지향프로그래밍\주차 실습\Project1\Debug\Project1.exe (프로세스 5560개)이 (가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] >
[디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```

예제 7-3: 이수영



```
1 #include <iostream>
2 using namespace std;
3 //7-3
4 class Rect;
5 class RectManager {
6 public:
7     bool equals(Rect r, Rect s);
8 };
9
10 class Rect {
11     int width, height;
12 public:
13     Rect(int width, int height) {
14         this->width = width; this->height = height;
15     }
16     friend RectManager; // RectManager클래스를 사용해야 하니 friend 키워드에 클래스명만 작성했습니다.
17 };
18
19 bool RectManager::equals(Rect r, Rect s) {
20     if (r.width == s.width && r.height == s.height)
21         return true;
22     else
23         return false;
24 }
25
26 int main() {
27     Rect a(3, 4), b(4, 5);
28     RectManager rm;
29     if (rm.equals(a, b))
30         cout << "equal" << endl;
31     else
32         cout << "not equal" << endl;
33 }
```

Microsoft Visual Studio 디버그 콘솔

```
not equal
C:\Users\User\source\repos\CppProject1030sy\Debug\CppProject1030sy.exe (프로세스 6952개)이(가) 종료되었습니다(코드: 0
개).
이 창을 닫으려면 아무 키나 누르세요...
```

16행의 friend RectManager;의 선언은 잘 했습니다. 클래스 모두 프렌드 함수로 쓰일 땐 매개변수 없이 클래스명 뒤에 세미콜론임.

그런데 7행 뒤에 void copy()를 끝까지 떠올리지 못했습니다. 클래스가 프렌드 함수일 때 copy() 기능이 필요한 지 몰랐다가 이후 복사가 필요하단 걸 알았습니다.

연산자 중복

- 원준서: C++언어에도 연산자 중복이 가능합니다. C++ 언어에 본래 부터 있던 연산자에 새로운 의미를 정의하고, 프로그램 가독성을 높여줍니다. 연산자 중복 사례 : +연산자. +연산자 중복의 사례로는 정수 더하기, 문자열 합치기, 색 섞기, 배열 합치기가 존재합니다.
- -C++에 본래 있는 연산자만 중복 가능합니다. -피 연산자 타입이 다른 새로운 연산을 정의합니다. -연산자는 함수 형태로 구현합니다. -연산자 함수(operator function) -반드시 클래스와 관계를 가집니다. -피연산자의 개수를 바꿀 수 없습니다. -연산의 우선 순위는 변경되지 않습니다. -모든 연산자가 중복 가능하지 않습니다.

연산자 중복

- 이수영: 연산자 중복은 이름은 같지만 기능 또는 매개변수의 수, 데이터타입이 다른 것을 뜻하며, 다형성을 보여줍니다. 예컨대 두 피연산자가 정수와 정수, 문자열과 문자열로 다를 때 연산자 중복이 성립되고, 객체지향의 특성 중 다형성이 잘 나타납니다. 특히 피연산자가 배열과 배열인 부분이 기억납니다. 두 배열이 결합한(merge), 원소들 간의 합집합으로 구성된 새로운 배열이 나타나는 점입니다. 1-6장의 다형성이 매개변수 데이터타입이 다른 것이었다면, 7장의 다형성은 피연산자 다른 것을 의미함을 새롭게 배웠습니다.

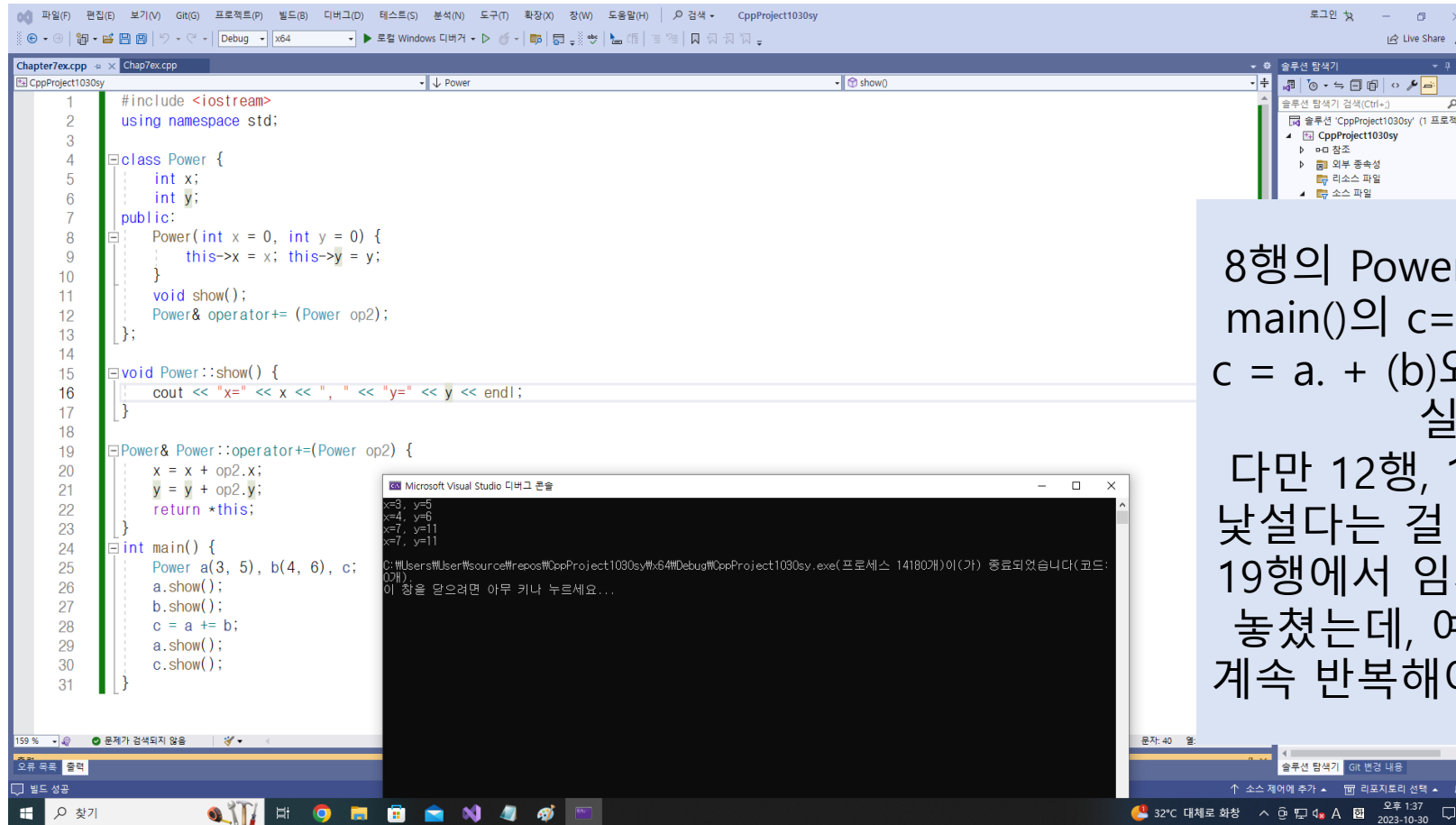
연산자를 함수(멤버/프렌드)로 표현

- 원준서: 연산자 함수를 구현하는 방법으로는 클래스의 멤버 함수로 구현하거나, 외부 함수로 구현하고 클래스에 프렌드 함수로 선언하는 두 가지 방법이 존재합니다.
- 연산자 함수 형식은 다음과 같습니다.
- 리턴타입 operator연산자(매개변수리스트);
- 이수영: 리턴타입 **operator** 연산자(매개변수리스트);
[예] **friend** bool **operator** == (**Color** op1, **Color** op2);

실습 과정에서 느낀 점: 원준서

- 예제 프렌드 함수 만들기, 다른 클래스의 멤버 함수를 프렌드로 선언, 다른 클래스 전체를 프렌드로 선언을 실습하는 과정에서 프렌드를 어떻게 선언해야 하는지 헷갈려 작성하는데 어려움이 있었습니다.
- 연산자 멤버 함수의 종류가 많다 보니, 특정 연산자 멤버 함수를 구현하는 부분에서 어려움이 있었습니다.

예제 7-4: 이수영



```
1 #include <iostream>
2 using namespace std;
3
4 class Power {
5     int x;
6     int y;
7 public:
8     Power(int x = 0, int y = 0) {
9         this->x = x; this->y = y;
10    }
11    void show();
12    Power& operator+=(Power op2);
13 };
14
15 void Power::show() {
16     cout << "x=" << x << ", " << "y=" << y << endl;
17 }
18
19 Power& Power::operator+=(Power op2) {
20     x = x + op2.x;
21     y = y + op2.y;
22     return *this;
23 }
24
25 int main() {
26     Power a(3, 5), b(4, 6), c;
27     a.show();
28     b.show();
29     c = a += b;
30     a.show();
31     c.show();
32 }
```

Microsoft Visual Studio 디버그 콘솔

```
x=3, y=5
x=4, y=6
x=7, y=11
x=7, y=11
이 창을 닫으려면 아무 키나 누르세요...
```

8행의 Power()는 디폴트 생성자.
main()의 c=a+b;인데 이 부분은
c = a. + (b)와 같고 이 때 12행이
실행됩니다.

다만 12행, 19행의 & 개념이 좀
낯설다는 걸 깨달았습니다. 이 때
19행에서 임시 객체 생성하는 걸
놓쳤는데, 예제 7-4는 손코딩을
계속 반복해야겠다고 느꼈습니다.

이항 연산자 중복

- 원준서: + 연산자

$c = a + b;$ 는 컴파일러에 의한 변환으로 $c = a . + (b);$ 로 변환됩니다.

- == 연산자 중복

$a == b$ 는 컴파일러에 의한 변환으로 $a . == (b)$ 로 변환됩니다.

- += 연산자 중복

$c = a += b;$ 는 컴파일러에 의한 변환으로 $c = a . += (b);$ 로 변환됩니다.

이항 연산자 중복

- 이수영: $a == b$ 는 $a. == (b)$ 로 변환되며, `bool operator == (Power op2);`는 b 가 $op2$ 에 전달됨을 의미합니다. 특히 `+=` 연산자 중복 부분이 기억에 남는데,

```
Power& Power::operator += (Power op2) {  
    kick = kick + op2.kick;  
    punch = punch + op2.punch;  
    return *this;  
}
```

`Power`의 참조가 리턴타입 이고, 실제 `return` 역시 나 자신의 주소를 옮겨 준다는 점이 기억에 남습니다. 즉, 나 자신의 복사본이 아니라 나 자신을 복사해서, $a += b$ 의 결과 a 와 c 의 결과가 같아집니다.

단항 연산자 중복

- 원준서: 단항 연산자는 피연산자가 하나 뿐인 연산자로, 연산자 중복 방식은 이항 연산자의 경우와 거의 유사합니다.
단항 연산자의 종류로는 전위 연산자(!op, ~op, ++op, --op), 후위 연산자(op++, op--)가 존재합니다.
- 전위 ++ 연산자 중복: ++a는 컴파일러에 의한 변환으로 a . ++ ()로 변환됩니다.
- 후위 연산자 중복, ++연산자: a++는 컴파일러에 의한 변환으로 a . ++ (임의의 정수)로 변환됩니다.

프렌드를 이용한 연산자 중복

- 원준서: cf) $2 + a$ 덧셈을 위한 $+$ 연산자 함수 작성

`power a(3,4), b;`

`b = 2 + a;`

`b = 2 + a;`는 컴파일러에 의한 변환으로 `b = + (2, a);`로 변환됩니다.

- [+연산자를 외부 프렌드 함수로 구현]

`c = a + b;`는 컴파일러에 의한 변환으로 `c = + (a, b);`로 변환됩니다.

- [단항 연산자 `++`를 프렌드로 작성하기]

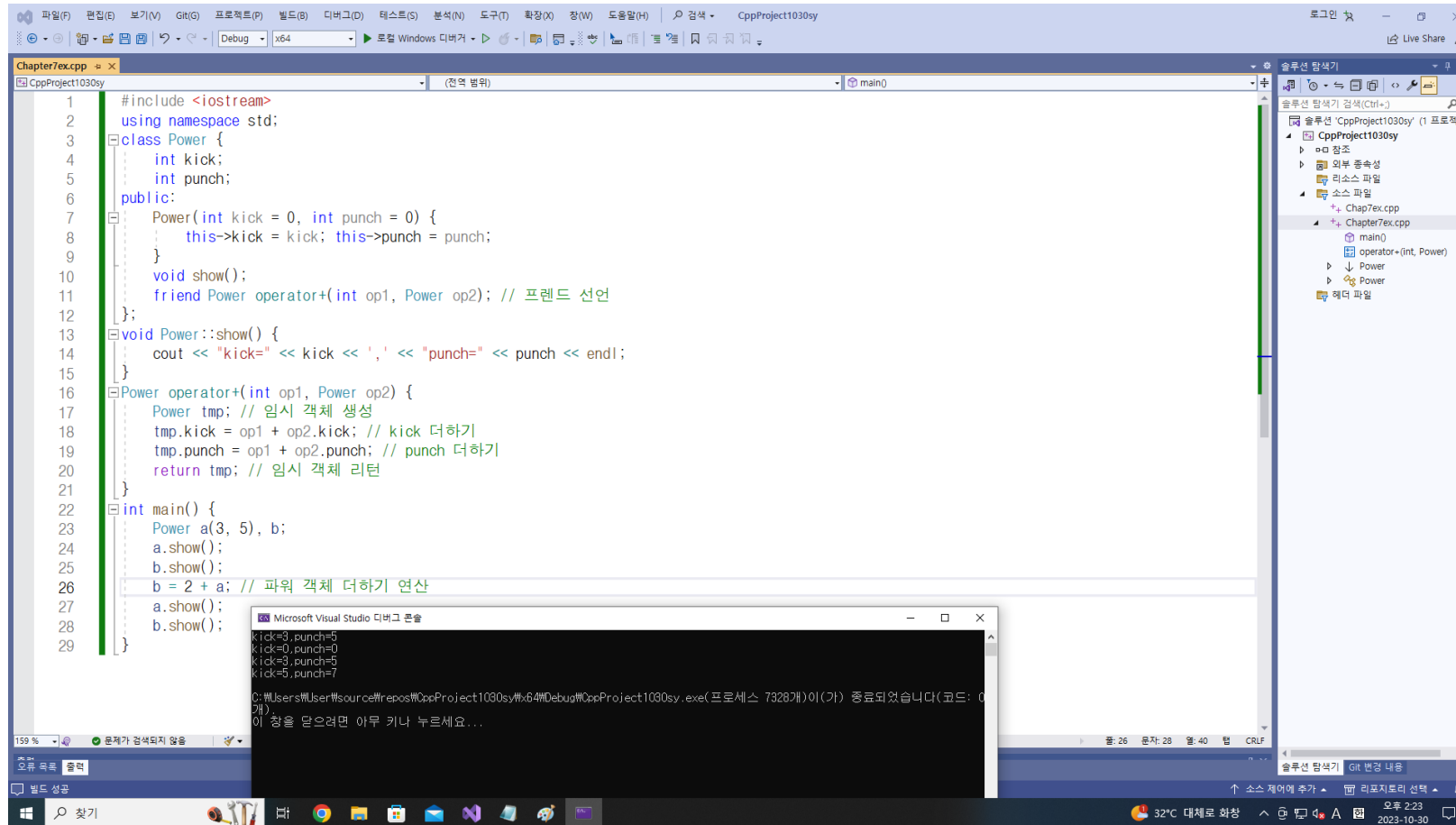
- 전위 연산자의 경우 `++a`는 컴파일러에 의한 변환으로 `++ (a)`로 변환됩니다.

- 후위 연산자의 경우 `a++`는 컴파일러에 의한 변환으로 `++ (a, 0)`로 변환됩니다.

단항 연산자 중복

- 이수영: 단항/다항 연산자는 피연산자의 개수가 1개/2개 이상인 것을 뜻하며, **전위** 연산자는 ++op, **후위** 연산자는 op++입니다.
- 전위 ++ 연산자 중복에 관해, ++a는 a. ++ ();로 변환됩니다.
Power& operator++ (); 매개변수는 없고, return *this; 입니다.
a=0, b=0일 때, b=++a라면, a = 1, b = 1입니다. 변화가 바로 반영되는 것입니다. **변경된 현재의 내가** 복사돼서 리턴됩니다.
- 반면, 후위 ++ 연산자 중복에서, a++는 a . ++ (정수)로 바뀌며,
Power operator ++ (int x); return tmp; 입니다. *this;가 아니며
a=0, b=0일 때, b=a++라면, a=1, b=0입니다. 변화가 반영되지 않습니다. **변경되기 이전 값이** 복사되어 리턴됐기 때문입니다.

예제 7-11 2+a;



The screenshot displays the Visual Studio IDE with a C++ project named 'CppProject1030sy'. The main file, 'Chapter7exc.cpp', contains the following code:

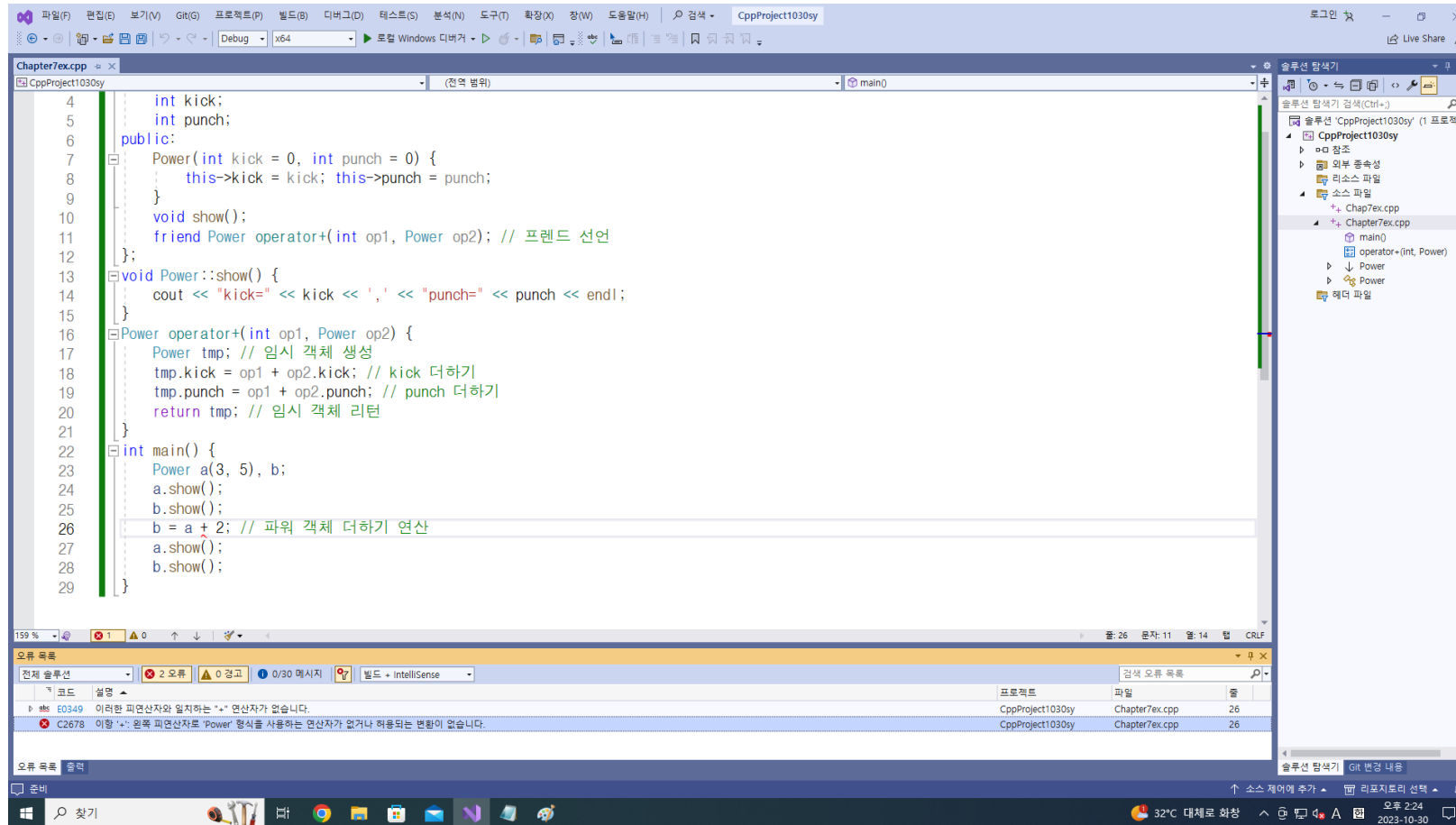
```
1 #include <iostream>
2 using namespace std;
3 class Power {
4     int kick;
5     int punch;
6 public:
7     Power(int kick = 0, int punch = 0) {
8         this->kick = kick; this->punch = punch;
9     }
10    void show();
11    friend Power operator+(int op1, Power op2); // 프렌드 선언
12 };
13 void Power::show() {
14     cout << "kick=" << kick << ', ' << "punch=" << punch << endl;
15 }
16 Power operator+(int op1, Power op2) {
17     Power tmp; // 임시 객체 생성
18     tmp.kick = op1 + op2.kick; // kick 더하기
19     tmp.punch = op1 + op2.punch; // punch 더하기
20     return tmp; // 임시 객체 리턴
21 }
22 int main() {
23     Power a(3, 5), b;
24     a.show();
25     b.show();
26     b = 2 + a; // 파워 객체 더하기 연산
27     a.show();
28     b.show();
29 }
```

The output window shows the results of the program execution:

```
kick=3, punch=5
kick=0, punch=0
kick=3, punch=5
kick=5, punch=7
```

The status bar at the bottom indicates the system temperature is 32°C and the time is 2:23 PM on 2023-10-30.

예제 7-11 a+2;



Pair Programming[옳은 3개, 틀린 것 2개]

(5)

```
Power ret_This() {  
    return this;  
}
```

: this는 주소를 가리키는 포인터. 그런데 Power는 객체다.
객체랑 주소값은 같을 수 없다. 객체를 리턴하고 싶은데
실제 리턴하는 건 주소값이라면 에러가 날 것이다.
따라서 (5)는 틀린 선지다.

Pair Programming[옳은 3개, 틀린 것 2개]

(4)

```
Power * ret_This() {  
    return this;  
}
```

: 반면, this는 동일하게 주소를 가리키는 포인터인데, Power *도 주소를 나타낸다. (5)는 주소값-객체로 맞지 않았는데, (4)는 객체-객체로 맞는다. 따라서 (4)는 옳은 선지다.

Pair Programming[옳은 3개, 틀린 것 2개]

(3)

```
Power& ret_This() {  
    return this;  
}
```

: 리턴이 this로 주소를 가리키므로 주소값이다. 그런데 Power&는 객체가 가리키는 주소인데, Power 객체의 '참조'와 this는 완전 다르므로, 같지 않다. 따라서 (3)은 틀린 선지다.

Pair Programming[옳은 3개, 틀린 것 2개]

(1)

```
Power& ret_This() {  
    return *this;  
}
```

: this는 포인터라 주소를 가리킨다. *this는 가리키는 값이 된다.
즉 Power&도 값, *this도 값이므로 (1)은 옳은 선지다.
(1)은 원래의 나를 넘겨주는 것이다.

Pair Programming[옳은 3개, 틀린 것 2개]

(2)

```
Power ret_This() {  
    return this;  
}
```

: Power 복사생성자는 객체다. 포인터가 가리키고 있는 내용은 this인데, 이걸 복사한 게 Power. 컴파일에러는 안 나겠지만, (2)는 **원래의 내가 아니라 나를 복사한 것을 넘겨주는 것이다.**

예제 7-14 참조를 리턴하는 << 연산자

- 이수영: main()에서 $a \ll 3 \ll 5 \ll 6$;를 쓸 수 있단 점이 기억납니다. **변경되기 이전 값**이 복사되는 것이 아니라, **변경된 현재 내가** 복사되는 점에 입각했기 때문입니다. 따라서 a 가 1일 때, $a \ll 3$ 의 결과는 변경되기 이전 값이 반영된 1 그대로가 아니라, $1+3$ 의 결과인 4입니다. 이 4에 $\ll 5$ 가 되면, 1이 아니라 $4+5$ 인 9입니다. 이 9에 $\ll 6$ 이 되면, 1이 아니라 $9+6$ 인 15입니다.
예제 7-14는 강의자료로 연습할 때는 단지 << 연산자를 배우는 것 같았는데, 실제 수업을 듣는 과정에서 앞에 배운 전위/후위 연산자의 차이를 다시 한 번 복습하면서, 값/주소에 의한 호출 역시 복습하는 과정이었기 때문에 가장 기억에 남습니다.