객체지향프로그래밍 정리노트(Week4)

202004029 김정호, 202004040 노성민

Q1. 프렌드 선언 3종류

7-1 프렌드 함수 만들기,
7-2 다른 클래스의 멤버 함수를 프렌드로 선언,
7-3 다른 클래스 전체를 프렌드로 선언

정호: 프렌드 함수는 클래스 외부에 선언하는 전역함수, 다른 클래스의 멤버 함수를 이용하는 함수, 다른 클래스의 모든 멤버 함수를 프렌드 함수로 선언하는 방법이 있는데 3가지 경우 모두 살펴보자. 먼저 전역함수를 보면 클래스 외부에 equals 함수를 선언해서 Rect 클래스의 friend 함수로 사용하고 있어. 따라서 equals 함수에서 Rect 클래스의 멤버 변수나 함수의 사용이 가능해지는 거야.

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);
};

bool equals(Rect r, Rect s) {
	if (r.width == s.width && r.height == r.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;
}
```

정호: 두 번째 예제는 RectManager 클래스의 equals 함수를 선언하여 그 함수를 Rect 클래스에서 friend 함수로 선언하여 사용하고 있어.

성민: 그러면 여기서도 RectManager 클래스의 equals 함수에서 Rect 클래스의 멤버를 전부 사용할 수 있겠구나,

7-2

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

class Rect;

class RectManager {
public:
	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 RectManager::equals(Rect r, Rect s);
};

bool RectManager::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);
	RectManager man;

	if (man.equals(a, b))
		cout << "equal" << endl;
	else
		cout << "not equal" << endl;
}
```

정호: 세 번째 예제는 RectManager 클래스 전체를 friend 함수로 받아서 RectManager의 모든 함수가 Rect 클래스의 멤버들의 접근할 수 있게 되었어.

7-3

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

class Rect;

class RectManager {
public:
	bool equals(Rect r, Rect s);
	void copy(Rect& dest, Rect& src);
};

class Rect {
	int width, height;
public:
	Rect(int width, int height) {
		this->width = width;
		this->height = height;
	}
	friend RectManager;
};

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

void RectManager::copy(Rect& dest, Rect& src) {
	dest.width = src.width;
	dest.height = src.height;
}

int main() {
	Rect a(3, 4), b(4, 5);
	RectManager man;

	man.copy(b, a);

	if (man.equals(a, b))
		cout << "equal" << endl;
	else
		cout << "not equal" << endl;
}
```

Q2. 연산자 함수

7-4 두 개의 Power 객체를 더하는 + 연산자 작성

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

class Power {
        int kick, punch;
public:
        Power(int kick = 0, int punch = 0) {
                this->kick = kick;
                this->punch = punch;
        }
        void show();
        Power operator + (Power op2);
};

void Power :: show() {
        cout << "kick = " << kick << " punch = " << punch << endl;
}

Power Power::operator+(Power op2) {
        Power tmp;
        tmp.kick = this->kick + op2.kick;
        tmp.punch = this->punch + op2.punch;
        return tmp;
}

int main() {
        Power a(3, 5), b(4, 6), c;
        c = a + b;
        a.show();
        b.show();
        c.show();
}
```

정호: 이 코드에서 중요한 부분은 파란색 부분이야. 임시 Power 객체인 tmp를 선언해서 두 Power 객체를 더한 결과를 저장할 목적으로 사용해서 tmp 객체의 kick, punch 멤버 변수에, 현재 객체(*this)의 kick, punch 멤버 변수와 op2 객체의 kick, punch 멤버 변수를 더한 값을 대입하는 거야. 그리고 최종적으로 두 Power 객체를 더한 결과인 tmp 객체를 반환하는거야. 여기서 주의해야 할 점은 반환 값을 주의해야 해.

7-6 두 개의 Power 객체를 더하는 += 연산자 작성

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

class Power {
	int kick, punch;
public:
	Power(int kick = 0, int punch = 0) {
		this->kick = kick;
		this->punch = punch;
	}
	void show();
	Power& operator += (Power op2);
};

void Power::show() {
	cout << "kick = " << kick << " punch = " << punch << endl;
}

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

int main() {
	Power a(3, 5), b(4, 6), c;
	a.show();
	b.show();
	c = a += b;
	a.show();
	c.show();
}
```

정호: 이 부분에 대해 한번 이번엔 성민이가 설명해보자.

성민: 이 함수는 Power 클래스의 객체에 대한 참조를 반환하는 함수야. 반환 형식이 Power&인 이유는 += 연산자를 사용한 후, 현재 객체 자체를 변경하고자 하기 때문이야. 따라서 함수의 반환 값은 현재 객체의 참조를 통해 업데이트된 값을 제공해야 해.

정호: 맞아 7-4 예제와 7-6 예제는 원하는 결과 값이 같지만 7-4 는 새로운 객체를 생성해 그 객체를 반환하는 것이고 7-6은 현재 객체에 대한 참조를 통해 새로운 객체를 만들지 않고 기존 객체의 정보를 수정해서 그 정보의 객체에 대한 참조를 반환하는 차이가 있어.

Q3 + 연산자 작성(실습): b = a + 2;

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

class Power {
	int kick, punch;
public:
	Power(int kick = 0, int punch = 0) {
		this->kick = kick;
		this->punch = punch;
	}
	void show();
	Power operator + (int op2);
};

void Power::show() {
	cout << "kick = " << kick << " punch = " << punch << endl;
}

Power Power::operator + (int op2) {
	Power tmp;
	tmp.kick = this->kick + op2;
	tmp.punch = this->punch + op2;
	return tmp;
}

int main() {
	Power a(3, 5), b(4, 6), c;
	a.show();
	b.show();
	c = a + 2;
	a.show();
	c.show();
}
```

성민: 7-4 예제처럼 이해하면 되겠어.

Q4. 예제 7-8 전위 ++ 연산자 작성

```
#include <iostream>
using namespace std;
class Power {
	int kick;
	int punch;
public:
	Power(int kick=0, int punch=0) {
	this->kick = kick; this->punch = punch;
}
void show();
	Power& operator++ (); // 전위 ++ 연산자 함수 선언
};
void Power::show() {
	cout << "kick=" << kick << ',' << "punch=" << punch << endl;
}
Power& Power::operator++() {
	kick++;
	punch++;
	return *this; // 변경된 객체 자신(객체 a)의 참조 리턴
}
int main() {
	Power a(3,5), b;
	a.show();
	b.show();
	b = ++a; // 전위 ++ 연산자 사용
	a.show();
	b.show();
}
```

정호: 이 부분도 현재 객체의 정보를 바꾸어 그 객체의 참조를 반환하여 전위 연산자를 구현했어,

Q5. 7-9(실습) Power 클래스에 ! 연산자 작성

```
#include <iostream>
using namespace std;
class Power {
	int kick;
	int punch;
public:
Power(int kick=0, int punch=0) {
	this->kick = kick;
	this->punch = punch;
```

```
}
        void show();
        bool operator! (); // ! 연산자 함수 선언
};
void Power::show() {
        cout << "kick=" << kick << ',' << "punch=" << punch << endl;
}
bool Power::operator!() {
        if(kick == 0 && punch == 0) return true;
        else return false;
}
int main() {
        Power a(0,0), b(5,5);
        if(!a) cout << "a의 파워가 0이다." << endl; // ! 연산자 호출
        else cout << "a의 파워가 0이 아니다." << endl;
        if(!b) cout << "b의 파워가 0이다." << endl; // ! 연산자 호출
        else cout << "b의 파워가 0이 아니다." << endl;
}
```

성민: 이 예제는 Power 객체의 kick, punch의 값을 비교하여 원하는 값을 파악한 것 같아. ! 연산자의 사용법을 보는 예제인거 같아.

Q6. 7-10 후위 ++ 연산자 작성

```
##include <iostream>
using namespace std;
class Power {
        int kick;
        int punch;
public:
Power(int kick=0, int punch=0) {
        this->kick = kick; this->punch = punch;
}
        void show();
        Power operator++ (int x); // 후위 ++ 연산자 함수 선언
};
void Power::show() {
        cout << "kick=" << kick << ','
                << "punch=" << punch << endl;
}
Power Power::operator++(int x) {
        Power tmp = *this; // 증가 이전 객체 상태를 저장
        kick++;
        punch++;
        return tmp; // 증가 이전 객체 상태 리턴
```

```
}
int main() {
	Power a(3,5), b;
	a.show();
	b.show();
	b = a++; // 후위 ++ 연산자 사용
	a.show(); // a의 파워는 1 증가됨
	b.show(); // b는 a가 증가되기 이전 상태를 가짐
}
```

정호: 이 예제는 새로운 객체의 현재 객체의 값을 저장하여 증가시켜주는 예제야. 이 과정에서 후위 연산자가 사용되어 a의 값은 증가 되지만 b는 a가 증가 되기 전에 이전 상태를 가지게 되는거지.

Q7. 7-11 2+a를 위한 + 연산자 함수를 프렌드로 작성

```
#include <iostream>
using namespace std;
class Power {
	int kick;
	int punch;
public:
Power(int kick=0, int punch=0) {
	this->kick = kick; this->punch = punch;
}
	void show();
	friend Power operator+(int op1, Power op2); // 프렌드 선언
};
void Power::show() {
	cout << "kick=" << kick << ',' << "punch=" << punch << endl;
}
Power operator+(int op1, Power op2) {
	Power tmp; // 임시 객체 생성
	tmp.kick = op1 + op2.kick; // kick 더하기
	tmp.punch = op1 + op2.punch; // punch 더하기
	return tmp; // 임시 객체 리턴
}
int main() {
	Power a(3,5), b;
	a.show();
	b.show();
	b = 2 + a; // 파워 객체 더하기 연산
	a.show();
	b.show();
}
```

성민: 여기서는 operator 함수를 외부함수로 선언하였기 때문에 friend문을 이용해서 Power의 멤버변

수에 접근할 수 있게 된거야.

정호: 맞아. 예제 7-12도 이와 마찬가지로 외부함수로 선언 뒤 friend문을 이용하여 멤버 변수에 접근하였고 다른 점이 있다면 매개변수를 2개 받아서 그 매개변수의 값을 더하는 방식이 차이가 있어.

Q8. 예제 7-13 ++연산자를 프렌드로 작성한 예

```
#include <iostream>
using namespace std;
class Power {
	int kick;
	int punch;
public:
	Power(int kick=0, int punch=0) { this->kick = kick; this->punch = punch; }
	void show();
	friend Power& operator++(Power& op); // 전위 ++ 연산자 함수 프렌드 선언
	friend Power operator++(Power& op, int x); // 후위 ++ 연산자 함수 프렌드 선언
};
void Power::show() {
	cout << "kick=" << kick << ',' << "punch=" << punch << endl;
}
Power& operator++(Power& op) { // 전위 ++ 연산자 함수 구현
	op.kick++;
	op.punch++;
	return op; // 연산 결과 리턴
}
Power operator++(Power& op, int x) { // 후위 ++ 연산자 함수 구현
	Power tmp = op; // 변경하기 전의 op 상태 저장
	op.kick++;
	op.punch++;
	return tmp; // 변경 이전의 op 리턴
}

int main() {
	Power a(3,5), b;
	b = ++a; // 전위 ++ 연산자
	a.show(); b.show();
	b = a++; // 후위 ++ 연산자
	a.show(); b.show();
}
```

정호: 여기서는 전위 연산자와 후위 연산자의 함수 구현이 중요해. 우선 전위 연산자는 참조를 반환하고 후위 연산자는 객체를 반환하는데 전위 연산자는 객체의 값들은 증가시켜야 하기때문에 참조를 반환하는 것이고 후위 연산자는 변경 이전의 값을 리턴하기 위해 이 와 같은 방식을 사용한 것 같아. 그리고 매개변수는 참조를 받아 값이 올바르게 증가 되도록 했어.

Q9. 예제 7-14 참조를 리턴하는 << 연산자 작성

```
#include <iostream>
using namespace std;
class Power {
	int kick;
	int punch;
public:
	Power(int kick = 0, int punch = 0) {
	this->kick = kick; this->punch = punch;
	}
	void show();
	Power& operator << (int n); // 연산 후 Power 객체의 참조 리턴
};
void Power::show() {
	cout << "kick=" << kick << ',' << "punch=" << punch << endl;
}
Power& Power::operator << (int n) {
	kick += n;
	punch += n;
	return *this; // 이 객체의 참조 리턴
}

int main() {
	Power a(1, 2);
	a << 3 << 5 << 6;
	a.show();
}
```

성민: 여기서는 위에 참조를 반환하는 형식과 동일하게 매개변수 int n의 값만큼 변수들에 더해 그 객체에 대한 참조를 반환하여 값을 증가시키는 방식을 사용하고있어.