Assignment #10

C++ Programming

부산대학교 정보컴퓨터공학부

201724444 김지원

Code 1. Constructor delegation, Constructor inheritance

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

class Computer
{
private:
    string cpu;
    int ram;
public:

    Computer(string _cpu)
        :Computer(_cpu,4) //Constructor delegation
    {
    }

    Computer(string _cpu, int _ram)
        :cpu(_cpu),ram(_ram)
    {
        cout << "cpu: " << cpu << "₩nram: " << ram << '₩n';
    }

};

class Notebook : public Computer
{
private:
    int battery;
public:
    using Computer::Computer; //super class의 생성자를 상속받음

    Notebook(string _cpu, int _ram, int _battery)
        :Computer(_cpu, _ram)
    {
        battery = _battery;
```

```
36          cout << "battery:  " << battery << "%\n";
37      }
38  };
39
40  int main(void)
41  {
42      cout << "--computer 1--\n";
43      Computer computer1("i3");
44      cout << "--computer 2--\n";
45      Computer computer2("i5", 8);
46      cout << "--notebook 1--\n";
47      Notebook notebook1("i7",16);
48      cout << "--notebook 2--\n";
49      Notebook notebook2("i9", 32, 80);
50
51      return 0;
52  }
```

- Constructor Delegation

  line 13 처럼 생성자가 다른 생성자를 사용하는 것을 말한다. 데이터 멤버가 늘어날수록 멤버 초기화 리스트에 멤버 수가 많아 질 것이다. 이를 해결하기 위해 주로 사용한다.

- Constructor Inheritance

  상속 생성자란 기본 클래스에 정의된 생성자를 상속받는 클래스에서 상속받을 수 있도록 하는 것이다. line 30 에서 using 선언을 이용하여 Computer class의 생성자를 상속받았다. 따라서 line 47과 같은 선언이 가능하다. 그러나 sub class에 추가된 data member에 대한 초기화가 잘 이루어지지 않으면 super class로부터 물려받은 생성자를 통해서만 객체를 생성하는 것은 위험하다.

output
--computer 1-- cpu: i3 ram: 4 --computer 2-- cpu: i5 ram: 8 --notebook 1-- cpu: i7 ram: 16 --notebook 2-- cpu: i9 ram: 32 battery: 80%

Code 2. Final virtual function, Final class, Virtual function override

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5
6  class Gun {
7  public:
8      Gun() {}
9      virtual void shoot() const = 0;
10 };
11
12 class LongGun : public Gun {
13 public:
14     virtual void shoot() const final override{ //final virtual function, virtual function override
15         cout << "Long gun Bang!₩n";
16     }
17 };
18
19 class ShortGun final : public Gun { //final class
20 public:
21     virtual void shoot() const  override{ //virtual function override
22         cout << "Short gun Bang!₩n";
23     }
24 };
25
26 int main(void)
27 {
28     Gun* long_gun = new LongGun();
29     Gun* short_gun = new ShortGun();
30     long_gun->shoot();
31     short_gun->shoot();
32     return 0;
33 }
```

output
Long gun Bang! Short gun Bang!

- Final virtual function

  특정 가상 함수가 derived class에서 재정의되는 것을 막기 위해 가상 함수 뒤에 final을 붙여준 것을 의미한다. line 14에서 final virtual function을 구현했다. 따라서 LongGun Class의 shoot()은 더 이상의 오버라이딩이 허용되지 않는다.

- Final class

  상속을 차단한 class이다. line 19에서 ShortGun class를 final로 선언했다. 따라서 다른 class에서 ShortGun class를 상속받지 받지 못하고, 상속 받으려고 하면 에러가 발생한다.

- Virtual function override

  line 9에 정의된 순수가상함수 shoot()을 line 14, 21에서 오버라이드(override)하고있다. 파생 클래스의 함수가 기반 클래스의 함수를 오버라이드 하기 위해서는 두 함수의 꼴이 정확히 같아야 한다. 또한 override 키워드를 통해 명시적으로 나타낼 수 있다. 이를 사용하게 되면 실수로 오버라이드 하지 않는 경우를 막을 수 있다.