TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 : Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 : 황수정
sue100012@naver.com
70 일차 (2018. 06. 08)

*class 가 필요한 이유
구조화 프로그래밍은 함수가 중심이고 객체 프로그래밍은 데이터가 중심이다.
*friend
class 내의 private 까지 접근 가능한 함수. class 간에 private 멤버 접근을 하기 위한 키워드이다.
접근하고자 하는 private 멤버를 갖는 클래스 내부에 friend 함수를 선언하면, 이곳 저곳의 class 멤버를 수정할 수 있다. 즉, A class 가 B class 대상으로 friend 선언을 하면 B class 는 A class 의 private 멤버에 직접 접근이 가능하다. 함수 뿐만 아니라 전역함수, 클래스, 클래스의 멤버함수, 변수에도 사용하여 똑같이 이용할 수 있다. 이 friend 선언은 어디에 위치하고 있어도 상관이 없다. 즉, friend 로 선언된 함수는 자신이 선언된 class 의 private 영역에 접근이 가능하다. 그러나, A class 도 B class 의 private 멤버에 접근하기 위해선 B class 가 A class 에게 friend 선언을 해줘야만 한다.

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

class Counter
{
   int val;                    //private 이 붙어 있어야 한다 원래..
public:
    Counter(void)
    {
        val = 0;
    }
    void Print(void)
    {
        cout << val <<endl;
    }
    friend void SetVal(Counter& c, int val);       //private 붙어 있어도 접근 가능하다. friend 이기 때문
};

void SetVal(Counter& c, int val)//위에서 friend 로 접근 가능하기 때문에 앞에 counter 가 안 붙은 것
{
   c.val = val;
}

int main (void)
{
   Counter cnt;
   cnt.Print();
   SetVal(cnt, 2002);
   cnt.Print();
   return 0;
}
```

결과 :
0
2002

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

class A             //A 입장에서는 B 가 friend 이다.
{
 private:
    int data;
    friend class B;
};

class B             //b 입장에서는 a 는 친구가 아님
{
 public:
    void SetData(A& a, int data)
    {
        a.data = data;
    }
};

int main(void)
{
  A a;
  B b;
  b.SetData(a, 10);
  return 0;
}
```

프린트하는 함수가 없어서 나오는 것은 없다. B 는 A 의 멤버를 건들일 수 있으나 그 반대로는 될 수 있다.
어느 입장에서 friend 가 선언되었는지가 중요하다.

[복사 생성자]

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

class A
{
 public:
    A(void)                 // 기본 생성자
    {
```

```
        cout << "A() Call" << endl;
    }
    A(int i)
    {
        cout << "A(int i) Call" << endl;
    }
    A(const A& a)                    //obj2 가 인자 객체가 복사할 때, 이런 식으로 된다.
    {
        cout << "A(const A& a) Call" << endl;
    }
};

int main(void)              //함수명이 같아도 인자가 다르면 다른 함수로 인식 된다.
{
  A obj1;
  A obj2(10);
  A obj3(obj2);  //객체 생성

  return 0;
}
```

결과

```
A() Call
A(int i) Call
A(const A& a) Call
```

복사 생성자 2 : p2 에 p1 을 복사. 즉, 생성자를 복사할 수 있다.

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

class Point
{
   int x, y;
 public:
    Point(int _x, int _y)
    {
        x = _x;
        y = _y;
    }
    void ShowData(void)
    {
        cout << x << ' ' << y <<endl;
    }
```

```
};

int main(void)
{
  Point p1(10, 20);
  Point p2(p1);              //생성자가 복사 된 것.
  p1.ShowData();
  p2.ShowData();
  return 0;
}
```
결과
10 20
10 20

```
#include <iostream>
#include <string.h>
using namespace std;

class Person
{
   char *name;
   char *phone;
 public:
   Person(char *_name, char *_phone);
   ~Person();
   void ShowData();
   Person(const Person&);
};

Person::Person(char *_name, char *_phone)
{
   name = new char[strlen(_name) + 1];
   strcpy(name, _name);

   phone = new char[strlen(_phone) + 1];
   strcpy(phone, _phone);
}

Person::~Person(void)
{
   delete []name;
   delete []phone;
```

```
}

Person::Person(const Person& p)
{
   name = new char[strlen(p.name) + 1];
   strcpy(name, p.name);

   phone = new char[strlen(p.phone) + 1];
   strcpy(phone, p.phone);
}

void Person::ShowData(void)
{
   cout << "name :" << name << endl;
   cout << "phone :" << phone << endl;
}

int main(void)
{
   Person p1("Jo", "011-9272-6523");
   Person p2 = p1;
   return 0;
}
```
출력하는 함수가 없어서 출력은 안 됨

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

class A
{
  int val;
 public:
   A(int i)
   {
      cout << "A(int i) Call" << endl;
      val = i;
   }
   A(const A& a)
   {
      cout << "A(const A& a) Call" << endl;
      val = a.val;
   }
```

```
    void ShowData(void)
    {A(int i) Call
A(const A& a) Call
val:30
        cout << "val:" << val << endl;
    }
};

void function(A a)
{
    a.ShowData();
}

int main(void)
{
    A obj(30);
    function(obj);
    return 0;
}
```

결과

```
A(int i) Call
A(const A& a) Call
val:30
```

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

class A
{
    int val;
public:
    A(int i)
    {
        cout << "A(int i) Call" << endl;
        val = i;
    }
    A(const A& a)
    {
        cout << "A(const A& a) Call" <<endl;
        val = a.val;
    }
```

```
    void ShowData(void)
    {
        cout << "val:" << val << endl;
    }
};

A function(void)
{
    A a(10);
    return a;
}

int main(void)
{
    function();
    return 0;
}
```

결과

```
A(int i) Call
```

변경된 코드

```
#include <iostream>

using namespace std;

class A
{
        int val;

public:
        A(int i)
        {
                cout << "A(int i) Call" << endl;
                val = i;
        }
        A(const A& a)
        {
                cout << "A(const A& a) Call" << endl;
                val = a.val;
        }
        void ShowData(void)
```

```
	{
		cout << "val : " << val << endl;
	}
};

A function(A& a)
{
	return a;
}

int main(void)
{
	A a(10);
	function(a).ShowData();
	return 0;
}
```

결과

```
A(int i) Call
A(const A& a) Call
val : 10
```

상속을 사용하는 이유?

사람(범주) – 이름, 나이 =학생, 군인,교수, 경찰 (직업)

같은 것을 다시 적을 필요 없는 것 - 재활용성

```
#include <iostream>
#include <string.h>
using namespace std;

class Person
{
   int age;
   char name[20];
 public:
   int GetAge(void) const
   {
	return age;
   }
   const char *GetName(void) const
   {
	return name;
   }
```

```
    Person(int _age = 1, char *_name = "noname")
    {
        age = _age;
        strcpy(name, _name);
    }
};

class Student: public Person
{
    char major[20];
 public:
    Student(char *_major)
    {
        strcpy(major, _major);
    }
    const char *GetMajor(void) const
    {
        return major;
    }
    void ShowData(void) const
    {
        cout << "name :" << GetName() << endl;
        cout << "age :" << GetAge() << endl;
        cout << "major:" << GetMajor() <<endl;
    }
};

int main(void)
{
    Student Park("Computer Science");
    Park.ShowData();
    return 0;
}
```

결과

```
name :noname
age :1
major:Computer Scienc
```