Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 및 회로 설계 전문가 과정

강사 – Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 – hoseong Lee(이호성)

hslee00001@naver.com

목차

1. friend

2. copy

3. copy_constructor

4. inheritance

5. operator

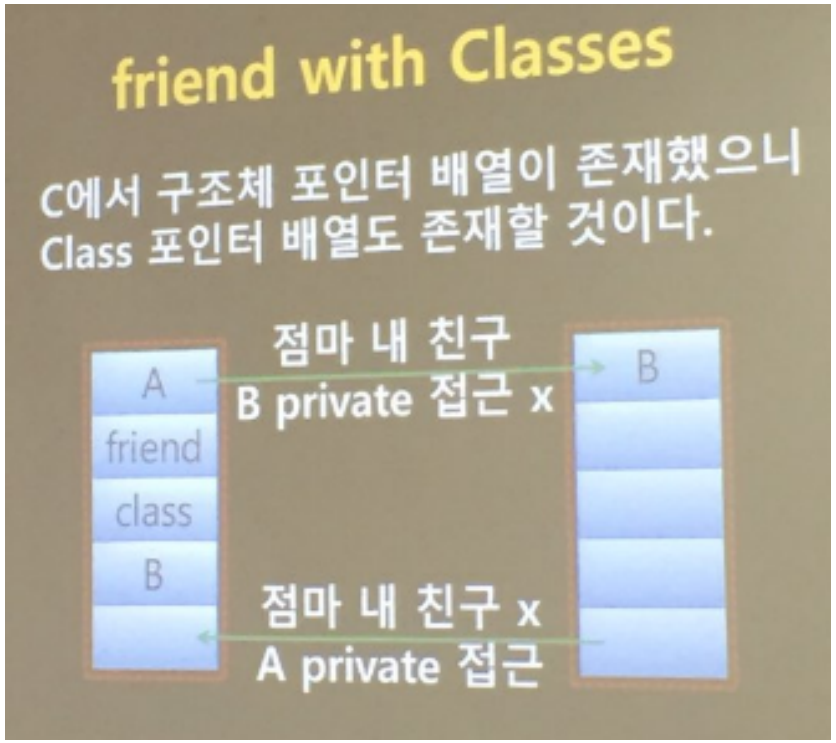
목차 control + 오른쪽 마우스 클릭

1. friend

```
lhs@lhs-Lenovo-YOGA-720-13IKB:~/my_proj/lec_code/6_8$ ./a.out
0
2002
lhs@lhs-Lenovo-YOGA-720-13IKB:~/my_proj/lec_code/6_8$ cat friend.cpp
#include <iostream>
using namespace std;

class Counter
{
        int val;
        public:
        Counter(void)
        {
                val = 0;
        }
        void Print(void)
        {
                cout << val << endl;
        }
        friend void SetVal(Counter &, int val);
};

void SetVal(Counter& c, int val)
{
        c.val = val;
}

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

public 위 val 변수 위에 private이 붙어야함..

SetVal함수는 friend이기 때문에 val에 접근이 가능함.



2.1 Copy

```
lhs@lhs-Lenovo-YOGA-720-13IKB:~/my_proj/lec_code/6_8$ ./a.out
A() Call
A(int i) Call
A(const A& a Call
lhs@lhs-Lenovo-YOGA-720-13IKB:~/my_proj/lec_code/6_8$ cat copy.cpp
#include <iostream>
using namespace std;

class A
{
public:
        A(void)
        {
                cout << "A() Call" << endl;
        }
        A(int i)
        {
                cout << "A(int i) Call" << endl;
        }
        A(const A& a)
        {
                cout << "A(const A& a Call" << endl;
        }
};

int main(void)
{
        A obj1;
        A obj2(10);
        A obj3(obj2);

        return 0;
}
```

2.2 copy 2

```
lhs@lhs-Lenovo-YOGA-720-13IKB:~/my_proj/lec_code/6_8$ ./a.out
10 20
10 20
lhs@lhs-Lenovo-YOGA-720-13IKB:~/my_proj/lec_code/6_8$ cat copy2.cpp
#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);

         p1.ShowData();
         p2.ShowData();
         return 0;
}
```

c++ 복사 생성자
copy construction 을 쓰는 이유

만약 p1을 p2 가 복사해서 값을 넣었다고 해보자. 그때 소멸자로 delete 한다면 p1 이 가르키는 값은 없게 된다. 이때 세그먼테이션 폴트가 난다.
이를 방지하기위해 copy 를 한다.

3.1 copy_construction

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

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

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("ho", "010-7307-8959");
    Person p2 = p1;

    return 0;
}
```

3.2 copy_constructon2

```
lhs@lhs-Lenovo-YOGA-720-13IKB:~/my_proj/lec_code/6_8$ ./a.out
A(int i) Call
A(const A&a) Call
val : 30
lhs@lhs-Lenovo-YOGA-720-13IKB:~/my_proj/lec_code/6_8$ cat copy_construct2.cpp
#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;
        }
};

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

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

3.3 copy_constructor3

```
lhs@lhs-Lenovo-YOGA-720-13IKB:~/my_proj/lec_code/6_8$ ./a.out
A(int i) Call
A(const A&a) Call
lhs@lhs-Lenovo-YOGA-720-13IKB:~/my_proj/lec_code/6_8$ cat copy_construct3.cpp
#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);
        return 0;
}
```

4. Inheritance

상속을 사용하는 이유? 1. 우선 상속엔 class재활용성, 2. 요구사항 변화에 따른 유연성

```cpp
#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   //Function Overloading 시 함수 뒤에 const를 붙임
    {
        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 << "magjor : " << GetMajor() << endl;
    }
};


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

public Person을 사용하면 Student 가 위 Person클래스를 모두 사용할 수 있다.

5.1 operator

연산자 오버로딩 (벡터, 행렬 , 복소수 연산을 단순화 할 수 있다.)

```
lhs@lhs-Lenovo-YOGA-720-13IKB:~/my_proj/lec_code/6_8$ ./a.out
3 4
13 14
lhs@lhs-Lenovo-YOGA-720-13IKB:~/my_proj/lec_code/6_8$ cat operator.cpp
#include <iostream>
using namespace std;

class Point
{
        private:
                int x,y;
        public:
                Point(int _x = 0, int _y = 0) : x(_x),y(_y) {}
                void ShowPosition(void);
                void operator + (int val);
};

void Point::ShowPosition(void)
{
        cout << x << " " << y << endl;
}

void Point::operator + (int val)
{
        x += val;
        y += val;
}

int main(void)
{
        Point p(3,4);
        p.ShowPosition();

        p.operator + (10);
        p.ShowPosition();
    return 0;
}
```

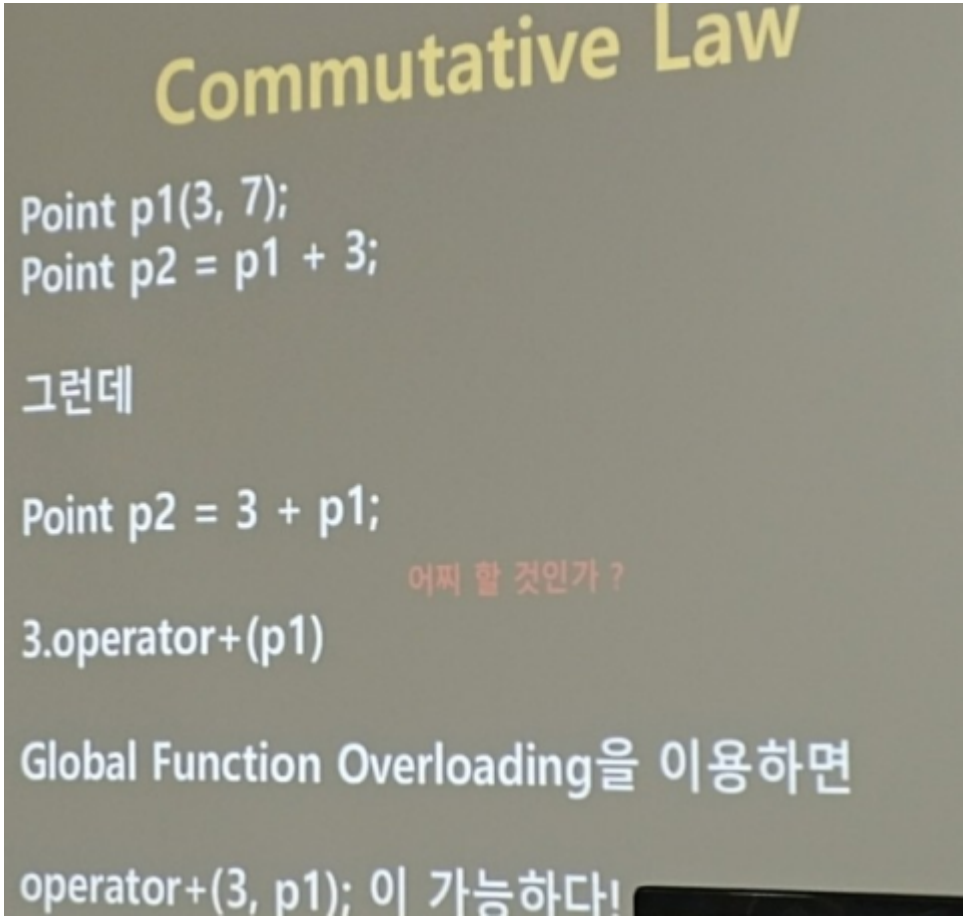
5.2 operator2

→ 벡터를 쉽게 표현할 수 있다.

```
lhs@lhs-Lenovo-YOGA-720-13IKB:~/my_proj/lec_code/6_8$ ./a.out
4 9
lhs@lhs-Lenovo-YOGA-720-13IKB:~/my_proj/lec_code/6_8$ cat operator2.cpp
#include <iostream>
using namespace std;

class Point
{
        private:
                int x,y;
        public:
                Point(int _x = 0, int _y = 0) : x(_x),y(_y) {}
                void ShowPosition(void);
                Point operator + (const Point& p);
};

void Point::ShowPosition(void)
{
        cout << x << " " << y << endl;
}

Point Point::operator+(const Point& p)
{
    Point temp(x + p.x, y + p.y);
        return temp;
}

int main(void)
{
        Point p1(1,2);
        Point p2(3,7);
        Point p3 = p1 + p2;
        p3.ShowPosition();
    return 0;
}
```

5.3 operator3

```
lhs@lhs-Lenovo-YOGA-720-13IKB:~/my_proj/lec_code/6_8$ ./a.out
3 7
4 8
34 78
34 78
lhs@lhs-Lenovo-YOGA-720-13IKB:~/my_proj/lec_code/6_8$ cat operator3.cpp
#include <iostream>
using namespace std;

class Point
{
        private:
                int x,y;
        public:
                Point(int _x = 0, int _y = 0) : x(_x),y(_y) {}
                void ShowPosition(void);
                Point& operator++(void);
                Point operator++(int);
};

void Point::ShowPosition(void)
{
        cout << x << " " << y << endl;
}

Point& Point::operator++(void)
{
    x++;
        y++;
        return *this;
}

Point Point::operator++(int)
{
        Point temp(x,y);
        //++(*this);  //이것은 요약한것. 지금우린 풀어씀.
        x++;
        y++;
        return temp;
}

int main(void)
{
        Point p1(3,7);
        (p1++).ShowPosition();
        p1.ShowPosition();

        Point p2(33,77);
        (++p2).ShowPosition();
        p2.ShowPosition();

    return 0;
}
```


Commutative Law
Point p1(3, 7);
Point p2 = p1 + 3;
그런데
Point p2 = 3 + p1;
어찌 할 것인가 ?
3.operator+(p1)
Global Function Overloading을 이용하면
operator+(3, p1); 이 가능하다!

5.4 operator4

```
lhs@lhs-Lenovo-YOGA-720-13IKB:~/my_proj/lec_code/6_8$ ./a.out
6 10
lhs@lhs-Lenovo-YOGA-720-13IKB:~/my_proj/lec_code/6_8$ cat operator4.cpp
#include <iostream>
using namespace std;

class Point
{
        private:
                int x,y;
        public:
                Point(int _x = 0, int _y = 0) : x(_x),y(_y) {}
                void ShowPosition(void);
                Point operator+(int val);
};

void Point::ShowPosition(void)
{
        cout << x << " " << y << endl;
}

Point Point::operator+(int val)
{
    Point temp(x + val, y + val);
        return temp;
}

int main(void)
{
        Point p1(3,7);
        Point p2 = p1 +3;
        p2.ShowPosition();

    return 0;
}
```

5.5 operator5

p1 + 3 은 됐는데, 3+ p1은 어떻게 할까..? 3은 우선 class도 아니다. 그렇다면 friend를 쓴다..

3.operator+(p) x

함수로 접근하면 operator + (숫자, 객체)

즉 ,순서에 따라 성능이 달라진다.

```
lhs@lhs-Lenovo-YOGA-720-13IKB:~/my_proj/lec_code/6_8$ ./a.out
6 10
13 17
lhs@lhs-Lenovo-YOGA-720-13IKB:~/my_proj/lec_code/6_8$ cat operator5.cpp
#include <iostream>
using namespace std;

class Point
{
        private:
                int x,y;
        public:
                Point(int _x = 0, int _y = 0) : x(_x),y(_y) {}
                void ShowPosition(void);
                Point operator+(int val);
                friend Point operator+(int val, Point& p);
};

void Point::ShowPosition(void)
{
        cout << x << " " << y << endl;
}

Point Point::operator+(int val)
{
    Point temp(x + val, y + val);
        return temp;
}

Point operator+(int val, Point& p)
{
        return p + val;
}

int main(void)
{
        Point p1(3,7);
        Point p2 = p1 +3;
        p2.ShowPosition();

        Point p3 = 7 + p2;
        p3.ShowPosition();
    return 0;
}
```