TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

2018-06-08 (70 회차)

강사 - Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 - 정유경
ucong@naver.com

1. friend with Classes
(C 에서 구조체 포인터 배열이 있었던 것처럼 C++은 Class 포인터 배열이 존재한다)

```
#include <iostream>
using namespace std;
class Counter{
	int val;
public:
	Counter(void){
		val =0;
	}
	void Print(void){
		cout << val << endl;
	}
	friend void SetVal(Counter&c, 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;
}
```

https://blog.naver.com/star7sss/220826110723

2. 한쪽만 Friend 인 경우

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

class A{
private:
	int data;
	friend class B;
};

class B{
public:
	void SetData(A& a, int data){
		a.data = data;
	}
};

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

Class B 는 A 의 친구이기 때문에
A 의 private int data 에 접근해서 변경이 가능하다.
B 는 A 가 friend 가 아니다.
따라서 B 가 A 에는 접근이 가능하지만 A 가 B 에는 접근할 수 없다.

3. 복사생성자 copy constructor

```
#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;
}
```

생성자가 3 개 있는 코드이다.
1. 기본생성자인 디폴트 생성자
2. 인자가 있는 생성자
3. 인자로 자신의 타입의 객체를 가지는 생성자
C 언어로 생각하면 자기 참조를 하는 구조체 A 로 볼 수 있다.
obj2 는 타입이 클래스이다
즉, 객체를 생성자로 받을 때는 const A&a 라는 형식으로 받는다

4. 복사생성자, 멤버 변수도 같이 복사하는 경우

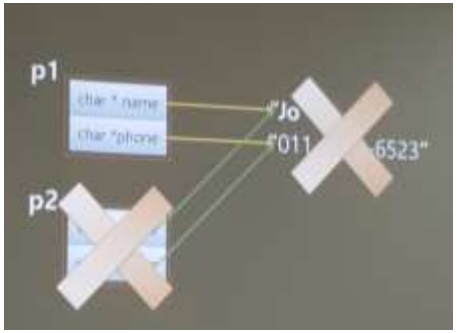
```
#include <iostream>
using namespace std;
class Point
{
	int x, y;
public:
	Point(int _x, int _y){
		x = _x;
		y = _y;
	}
	void ShowData(void){
		cout << x << ' ' << y <<endl;
	}
};

int main(void){
	Point p1(10, 20);
	Point p2(p1); // p2 에  p1 즉 10, 20 이 복사된다.
	p1.ShowData();
	p2.ShowData();
	return 0;
}
```

http://thrillfighter.tistory.com/146
http://metalkim.tistory.com/245

5. Segmentation Fault 를 피하기 위한 방법으로
복사생성자를 사용한다.

```
#include <string.h> // 추가할 것
#include <iostream>
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("Jo", "011-9272-6523");
	Person p2 = p1;
	//p1.ShowData();
	//p2.ShowData();
	return 0;
}
```

*. Person::Person(const Person& p) 부분을 추가해 주어야
같은 메모리 공간을 p1, p2 가 동시에 가리키지 않게
되어 Segmentation Fault 를 막을 수 있다.
즉, 다음과 같은 상황을 막을 수 있다.



*. Destructor 는 return 0 시 바로 동작함
https://kin.naver.com/qna/detail.nhn?d1id=1&dirId=1040101&docId=64156825&qb=c2VnbWVudGF0aW9uIGZhdWx0IOuzteyCrOyDneyEseyekA==&enc=utf8&section=kin&rank=1&search_sort=0&spq=0&pid=TxFNPspySoCsss5pSC8ssssstnd-033954&sid=z246APp/FwnQPF8jxENsNA%3D%3D
*. p1.~[소멸자] 하여 호출하고
p1.showdata 하면 출력되지 않는다.

그렇다면 소멸자 호출 시점은 언제일까
http://mystyle1057.tistory.com/56

6. 복사생성자 동작 – 두번째 케이스

*. 복사생성자(Copy Constructor)는 언제 동작하는가

1. 기존에 생성된 객체로 새로운 객체를 초기화 할 때

2. 함수 호출 시 객체를 Reference 가 아닌 형태로 전달할 경우

3. 함수 내에서 객체를 Reference 가 아닌 형태로 return 하는 경우

```
#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;
}
```

*. function(obj) 하는순간 두번째 복사생성자가 실행된다

→ 두번째 케이스 예제

→ A(int i) Call 나오고 복사생성자 실행되면서 function 에 의해 show data 함수(함수실행)가 튀어나온다

A(int i) Call

A(const A& a) Call

val:30

*. 연산자 오버로딩에서 가장 많이 쓰는게 복사 생성자이다

7. 복사생성자 동작 - 세번째 케이스

```
#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;
}
```

→ A a(10) 디폴트 생성자 에 의해 A(int I){} 가 실행된다.

http://blog.eairship.kr/173

코드를 다음과 같이 변경하여 실행해본다.

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

→ A a(10) 디폴트 생성자 에 의해 A(int I){} 가 실행되고,

return a; 에 의해 세번째 케이스로써 복사 생성자가 실행된다

.

8. 상속

```
#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 Jamie("Embedded System");
        Jamie.ShowData();
        return 0;
}
```

상속을 사용하는 이유는

1. 사람이라는 범주에서 이름, 나이라는 두가지 항목을 사용한다고 할 때, 학생 , 군인, 교수, 경찰에 대해

항목들을 입력하는 경우에 각각 입력하기 번거롭다.

따라서 사람이라는 클래스를 상속받는다. (Class 재활용성)

2. 나이 대신 멘탈이라는 항목을 사용하고 싶을때 나이를 멘달로 바꾸어 상속시킨다(요구사항 변화에 따른 유연성)

http://blog.eairship.kr/173

*. main 에서 name , age 정보가 없는데 22 번째 줄에 디폴트 파라미터가 들어가 있다.

*. class Student: public Person

person 에 있는 모든 정보를 student 가 사용할 수 있다고 하는 것이 바로 상속의 개념이다.

9. 연산자 오버로딩 : 새로운 연산 규칙을 만든다. 3+3 = 9

```
#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();
	return 0;
}
```

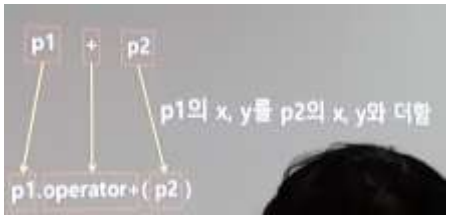
*. x 에 _x 를 집어 넣겠다는 뜻

*. x, y 에 3,4 가 셋팅된다.

*. Show position 하면 3,4 나오고

operator + 로 연산자를 재정의 하면

13,14 가 출력된다.



10. 위의 코드는 수동적인 느낌이라면 이번에는 자동적으로

```
#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; // 4 9 나온다
	p3.ShowPosition();
	return 0;
}
```

*. point p3 = p1 + p2;

C 언어에서 구조체 변수 2 개를 더하는 작업은 할 수 없는데

C++에서는 두개를 더할 수 있다

따라서 벡터의 덧셈을 쉽게 구현할 수 있다.

P1 +3 일 경우에도 복사연산자는 동작하지만

3+p1 은 동작하지 않는다. 3 은 클래스가 아니므로

11. 단항연산자 – 전위, 후위연산 구별은 전달인자로 한다

```
#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& operator--(int){
	Point temp(x,y);
	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;
}
```

https://blog.naver.com/kse5217/221214482196

연산자 오버로딩에서 전위 후위 연산자를 어떻게 구별할까

++p 와 p++를 구별하기 위한 규칙을 만들었다.

전위연산이면 인자로 void 를, Point opeartor++(void)

후위이면 int 를 준다.

12. friend 는 예외이다. - - 자체가 하나의 함수가 된다.

```
#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);
```

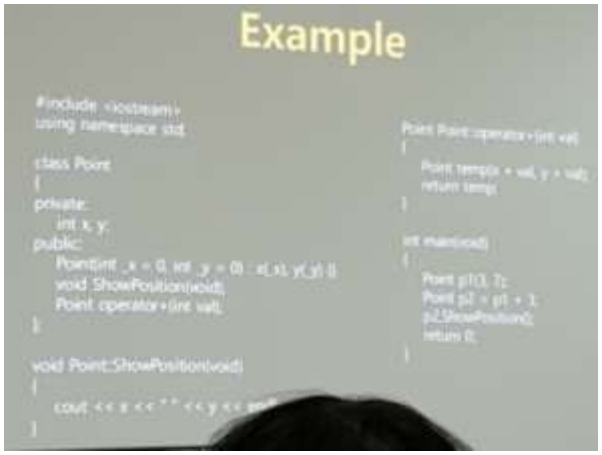
```
        Point& operator++(void);
        friend Point& operator--(Point& p);
};
void Point::ShowPosition(void){
        cout << x << " " << y << endl;
}
Point& Point::operator++(void){
        x++;
        y++;
        return *this;
}
Point& operator--(Point& p){
        p.x--;
        p.y--;
        return p;
}

int main(void){
        Point p1(3,7);
        ++p;
        p.ShowPosition();
        --p;
        p.ShowPosition();
        ++(++p);
        p.ShowPosition();
        --(--p);
        p.ShowPosition();
        return 0;
}
```

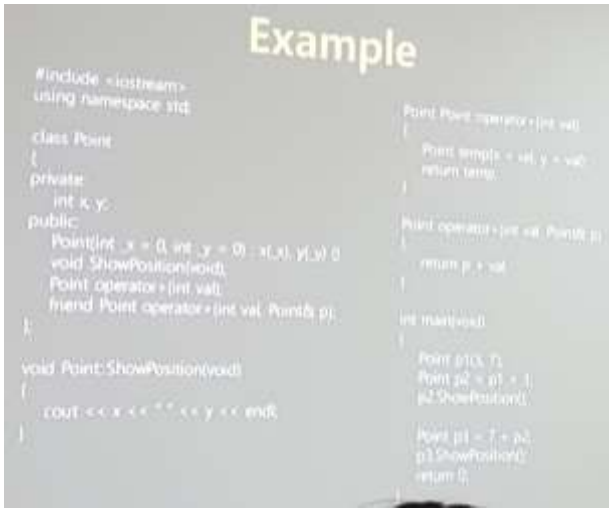
*. friend 가 필요할 수도 필요하지 않을 수 도 있다.

13. 3+p 는 방법 없다

3+operator+(p) 를 friend 가 해준다

아래 두개의 코드를 비교해보자





http://showmiso.tistory.com/32

http://algamza.blogspot.com/2016/03/c-operator-overloading.html

14. **템플릿**(template): 함수나 클래스가 개별적으로 다시 작성하지 않고도 각기 다른 수많은 자료형에서 동작할 수 있게 한다. 즉, 타입을 알아서 지정해준다.

```
#include <iostream>

using namespace std;

template <typename T> // 타입 이름이 T 이다


T Add(T a, T b){

        return a+b;

}


int main(void){

        cout << Add(10,20) << endl;

        cout << Add(1.1, 2.2) << endl;


// 문자 두개를 더하는건 다음과 같이 해준다.

        printf("%d₩n", Add('a', '1'));

// 출력서식을 10 진수 숫자로 지정해주기 위해 %d 사용
        return 0;

}
```

*. AM5728 Hetero Arch > 교재 > [9] ~ [11] 진행함

http://cafe.naver.com/hestit/2055

*. ~/ti-processor-sdk-linux-am57xx-evm-04.03.00.05$ ./setup.sh 실행

→ Which serial port do you want to use with minicom?

[/dev/ttyS0] /dev/ttyusb

minicom 나오면 /dev/ttyusb 지정해준다.

→ SD Card 선택

→ 보드 연결하라는 문구에서 ^c

→ 설치 끝나면

SD 카드 뽑아서 DSP 에 넣는다