21장. 클래스와 객체

01_ 클래스와 객체의 기본
02_ 클래스와 객체 자세히 살펴보기

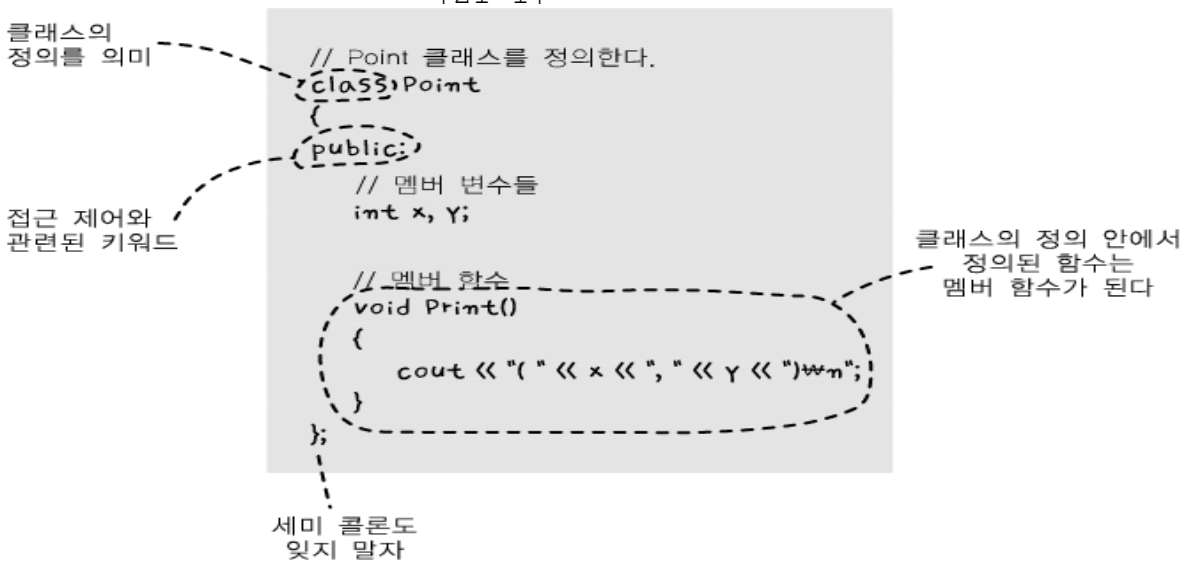
박 종 혁 교수
UCS Lab
Tel: 970-6702
Email: jhpark1@seoultech.ac.kr

클래스의 정의

- Point 클래스를 정의하는 예

```
// Point 클래스를 정의한다.
class Point
{
public:
        // 멤버 변수들
        int x, y;

        // 멤버 함수
        void Print()
        {
                cout << "( " << x << ", " << y << ")\n";
        }
};
```

[21-1]



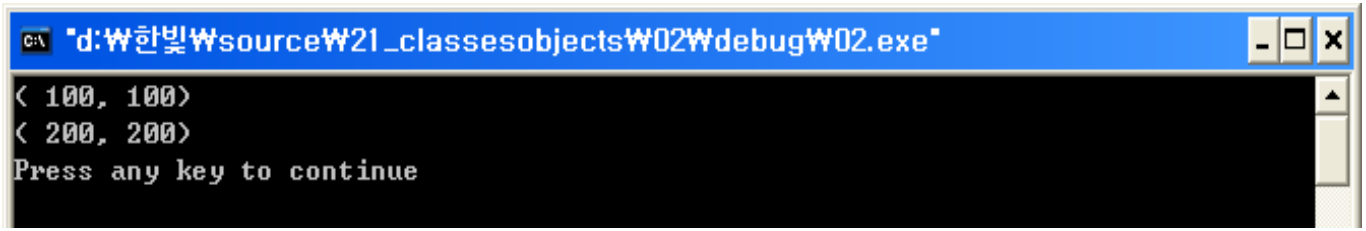
객체의 생성과 사용(1)

- 클래스 객체를 생성하고 사용하는 예

```
// 객체를 생성한다.
Point pt1, pt2;

// pt1, pt2를 초기화 한다.
pt1.x = 100;
pt1.y = 100;
pt2.x = 200;
pt2.y = 200;

// pt1, p2의 내용을 출력한다.
pt1.Print();
pt2.Print();
```

[21-2]

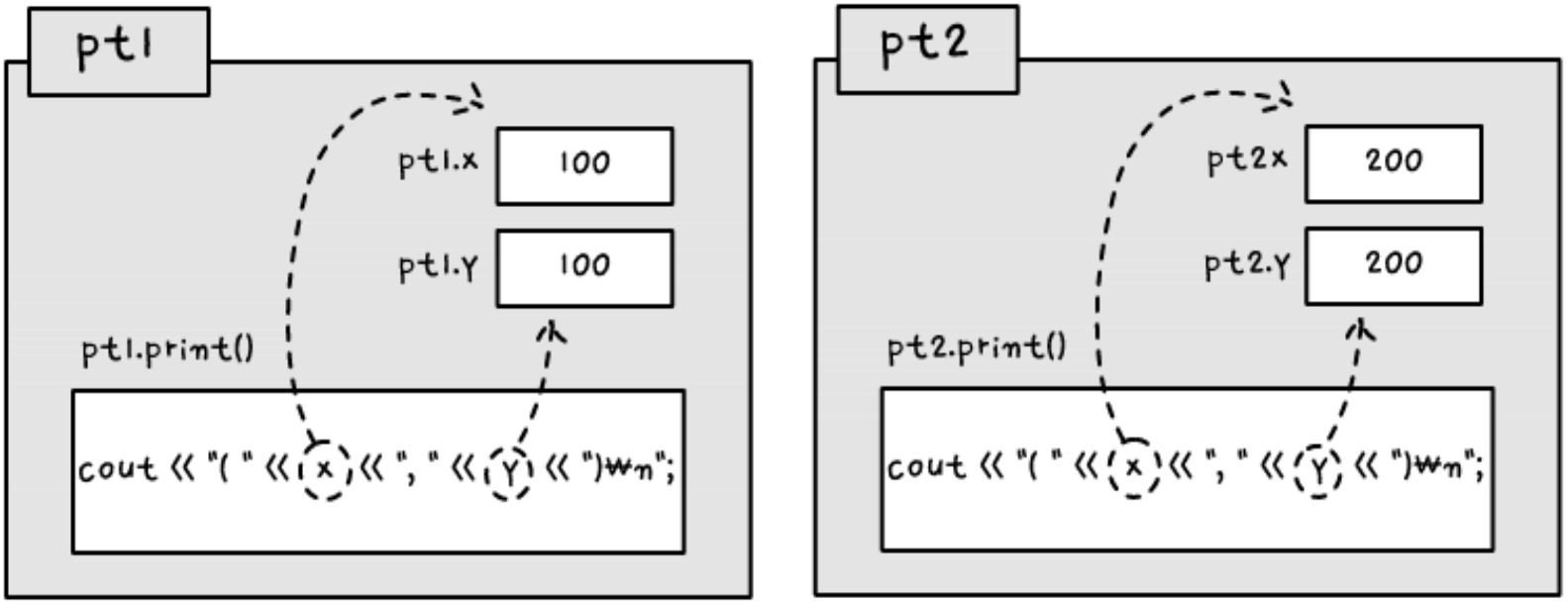
- 실행 결과



객체의 생성과 사용(2)

- Print() 함수에서 사용하는 x, y의 의미

[21-3]



일반 함수와 멤버 함수의 차이점

- 다음과 같은 차이점을 가지고 있다.
  - 멤버 함수를 호출하기 위해서는 객체의 이름을 명시해주어야 한다.
  - 멤버 한수 안에서는 객체의 이름을 명시하지 않고 멤버에 접근할 수 있다.
  - 외부에서 접근 할 수 없도록 설정된 멤버라 할지라도 멤버 함수 안에서는 접근할 수 있다.

멤버 함수의 위치

- 클래스의 정의 바깥쪽에 멤버 함수를 정의한 예

```
class Point
{
public:
          // 멤버 변수
          int x, y;

          // 멤버 함수
          void Print();
};

void Point::Print()
{
          cout << "( " << x << ", " << y << ")\n";
}
```

멤버 함수 안에서의 이름 충돌

- 클래스의 정의 바깥쪽에 멤버 함수를 정의한 예

```
class Point
{
public:
          // 멤버 변수
          int x, y;

          // 멤버 함수
          void Print();
};

// 멤버 함수
void Point::Print()
{
          int x = 333;
          cout << "( " << x << ", " << y << ")\n";
}
```

[21-4]

- 실행 결과

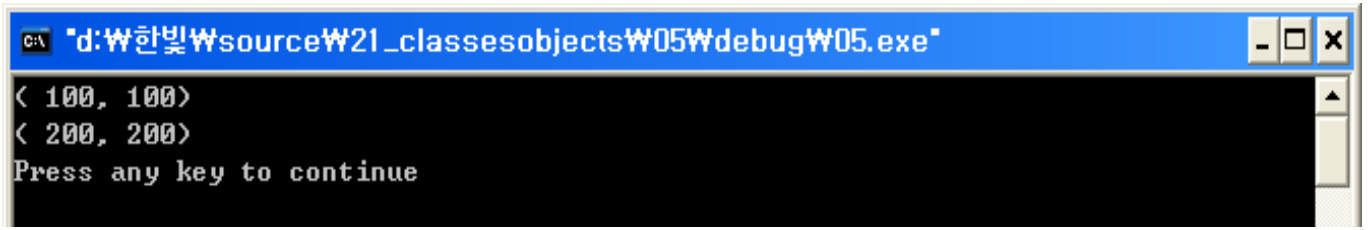
객체를 사용한 초기화와 대입

- 클래스의 정의 바깥쪽에 멤버 함수를 정의한 예

```
Point pt1, pt2;
pt1.x = 100;
pt1.y = 100;
pt2.x = 200;
pt2.y = 200;

// pt1을 사용해서 새로운 pt3를 초기화 한다.
Point pt3 = pt1;

pt3.Print();

// pt2을 pt3에 대입한다.
pt3 = pt2;

pt3.Print();
```

[21-5]

- 실행 결과



생성자와 소멸자

- 생성자는 객체를 생성할 때 자동으로 호출되는 함수이다.
  - 그러므로 생성자는 객체를 사용할 수 있도록 초기화 하는 코드를 넣기에 알맞은 장소이다.

- 소멸자는 객체를 소멸할 때 자동으로 호출되는 함수이다.
  - 그러므로 소멸자는 객체가 사용한 리소스를 정리하는 코드를 넣기에 알맞은 장소이다.

디폴트 생성자(Default Constructors)

- 디폴트 생성자의 추가

```
class Point
{
public:
          int x, y;
          void Print();
          Point();
};

Point::Point()
{
          x = 0;
          y = 0;
}

// 중간 생략

Point pt;            // 생성자가 호출된다.
pt.Print();
```

[21-6]

- 실행 결과

인자가 있는 생성자(1)

- 인자가 있는 생성자의 추가

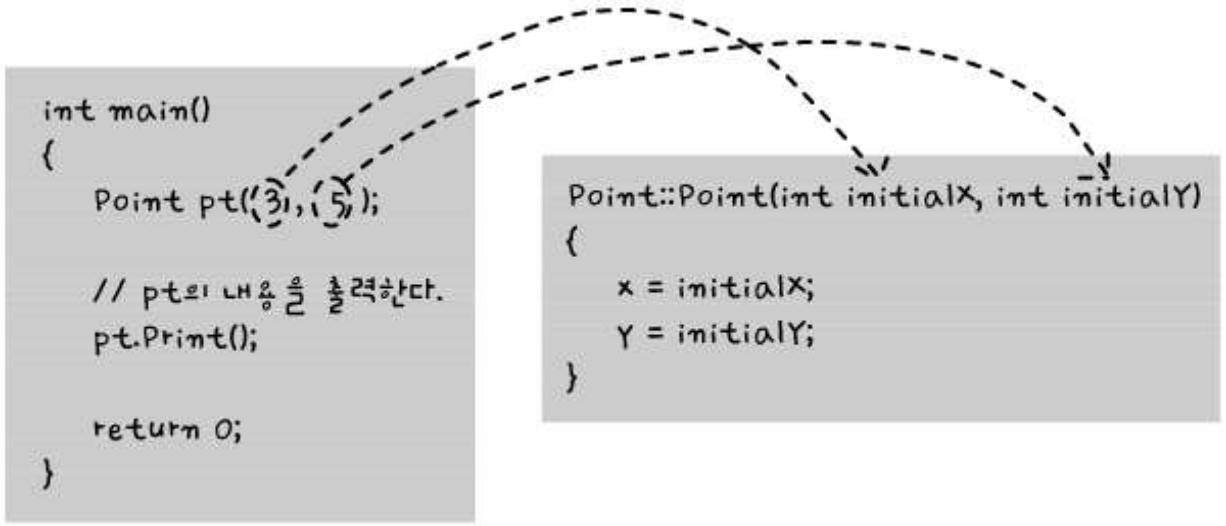
```
class Point
{
public:
          int x, y;
          void Print();
          Point();
          Point(int initialX, int initialY);

};

Point::Point(int initialX, int initialY)
{
          x = initialX;
          y = initialY;
}

// 중간 생략

Point pt(3, 5);
pt.Print();
```

인자가 있는 생성자(2)

- 실행 결과

[21-8]

- 생성자로의 인자 전달



복사생성자(Copy Contructors)(1)

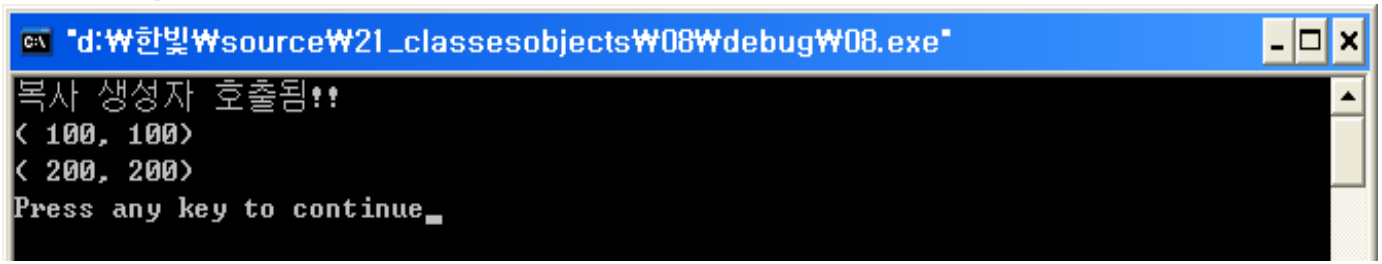
- 복사 생성자의 추가

[21-8]

```
class Point
{
public:
          int x, y;
          void Print();
          Point();
          Point(int initialX, int initialY);
          Point(const Point& pt);
};

Point::Point(const Point& pt)
{
          cout << "복사 생성자 호출됨!!\n";
          x = pt.x;
          y = pt.y;
}

// 중간 생략

Point pt1(100, 100), pt2(200, 200);

// pt1을 사용해서 새로운 pt3를 초기화 한다.
Point pt3 = pt1;
pt3.Print();

// pt2을 pt3에 대입한다.
pt3 = pt2;
pt3.Print();
```

복사생성자(Copy Contructors)(2)

- 실행 결과



[21-9]

- 복사 생성자는 자기 자신의 타입에 대한 레퍼런스를 인자로 받는 생성자다. 그러므로 다음의 두 가지 원형 중 한 가지 모습을 가질 수 있다.

```
Point( Point& pt );
Point( const Point& pt );
```

- 다음과 같은 두 가지 방식으로 복사 생성자를 호출할 수 있다.

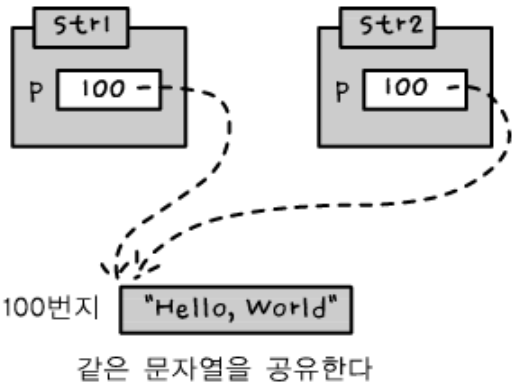
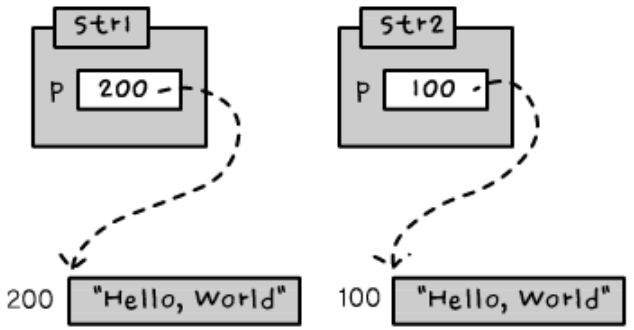
```
Point pt1;
Point pt2 = pt1;
Point pt3( pt1 );
```

얕은 복사와 깊은 복사

- 기본적으로 제공되는 복사 생성자는 얕은 복사를 하도록 구현되어 있다. 깊은 복사가 필요하다면 별도의 복사 생성자를 만들어야 한다.

	얕은 복사(Shallow Copy)	깊은 복사(Deep Copy)

복사 생성자	// 복사 생성자가 단순히 1:1 복사만 하는 경우
	String::String(const String& s)
	{
	p = s.p;
	}
		// 복사 생성자가 문자열을 복사해주는 경우
		String::String(const String& s)
		{
		p = new char [s.size() + 1];
		strcpy(p, s.p);
		}

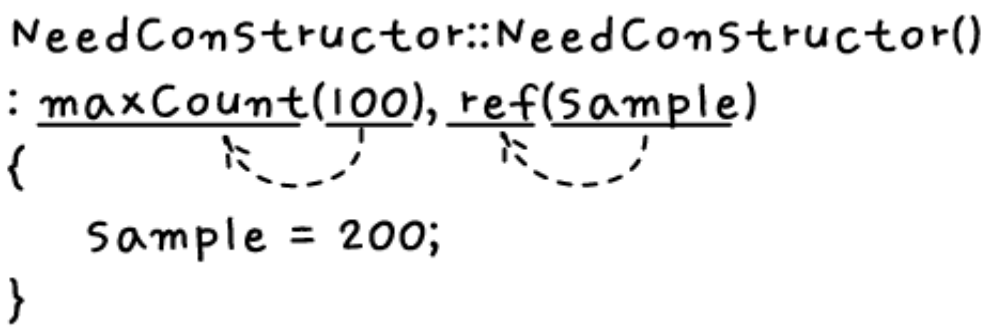
// 다음 문장의 결과 String str1 = str2;	 같은 문자열을 공유한다	 서로 다른 문자열을 가지고 있다

반드시 생성자가 필요한 경우

- 멤버 변수중에 레퍼런스 변수 혹은 Const 속성의 변수가 있는 경우에는 반드시 생성자에서 초기화 해주어야 한다.

```
class NeedConstructor
{
public:
          const int maxCount;
          int& ref;
          int sample;

          NeedConstructor();
};

NeedConstructor::NeedConstructor()
: maxCount(100), ref(sample)
{
          sample = 200;
}
```

[21-15]



소멸자(1)

- 소멸자를 사용해서 할당한 메모리를 해제하는 예

```
class DynamicArray
{
public:
          int* arr;
          DynamicArray(int arraySize);
          ~DynamicArray();
};

DynamicArray::DynamicArray(int arraySize)
{
          // 동적으로 메모리를 할당한다.
          arr = new int [arraySize];
}

DynamicArray::~DynamicArray()
{
          // 메모리를 해제한다.
          delete[] arr;
          arr = NULL;
}
```

소멸자(2)

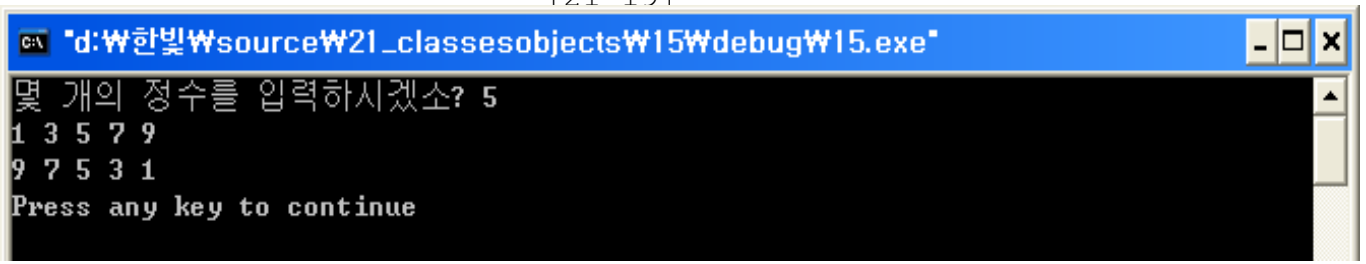
- 소멸자를 사용해서 할당한 메모리를 해제하는 예

```
int main()
{
        // 몇 개의 정수를 입력할지 물어본다.
        int size;
        cout << "몇 개의 정수를 입력하시겠소? ";
        cin >> size;

        // 필요한 만큼의 메모리를 준비한다.
        DynamicArray da(size);

        // 정수를 입력 받는다.
        for (int i = 0; i < size; ++i)
                cin >> da.arr[i];

        // 역순으로 정수를 출력한다.
        for (i = size - 1; i >= 0; --i)
                cout << da.arr[i] << " ";

        cout << "\n";

        // 따로 메모리를 해제해 줄 필요가 없다.

        return 0;
}
```

소멸자(3)

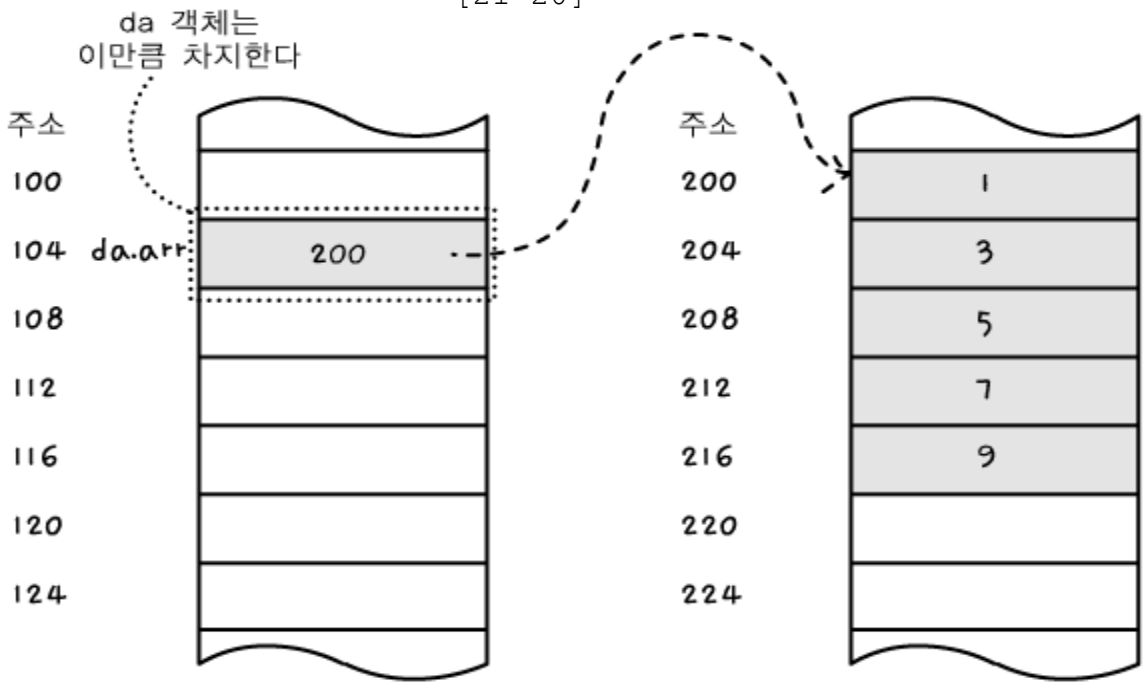
- 실행 결과

[21-19]



- DynamicArray 객체를 생성한 모습

[21-20]



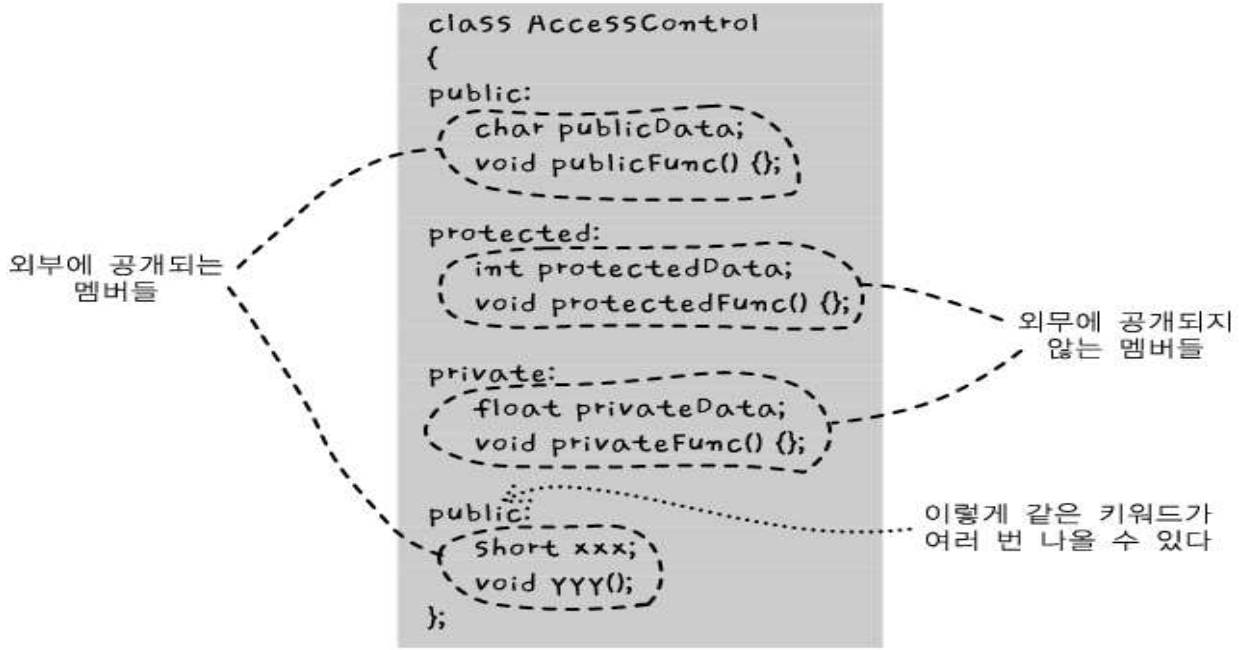
접근 권한 설정하기(1)

- 멤버의 접근 권한을 설정하는 예

```
class AccessControl
{
public:
          char publicData;
          void publicFunc() {};
protected:
          int protectedData;
          void protectedFunc() {};
private:
          float privateData;
          void privateFunc() {};
};

int main()
{
          // 객체를 생성하고, 각 멤버에 접근해보자
          AccessControl ac;

          ac.publicData = 'A';          // 성공
          ac.publicFunc();              // 성공
          ac.protectedData = 100;       // 실패
          ac.protectedFunc();           // 실패
          ac.privateData = 4.5f;        // 실패
          ac.privateFunc();             // 실패

          return 0;
}
```

접근 권한 설정하기(2)

- 멤버의 접근 권한 설정하기



[21-23]

- 접근 권한 키워드에 대한 요약 (뒤에서 더욱 자세히 분류)
  - public : 외부에서의 접근을 허용한다.
  - protected, private : 외부에서 접근할 수 없다.

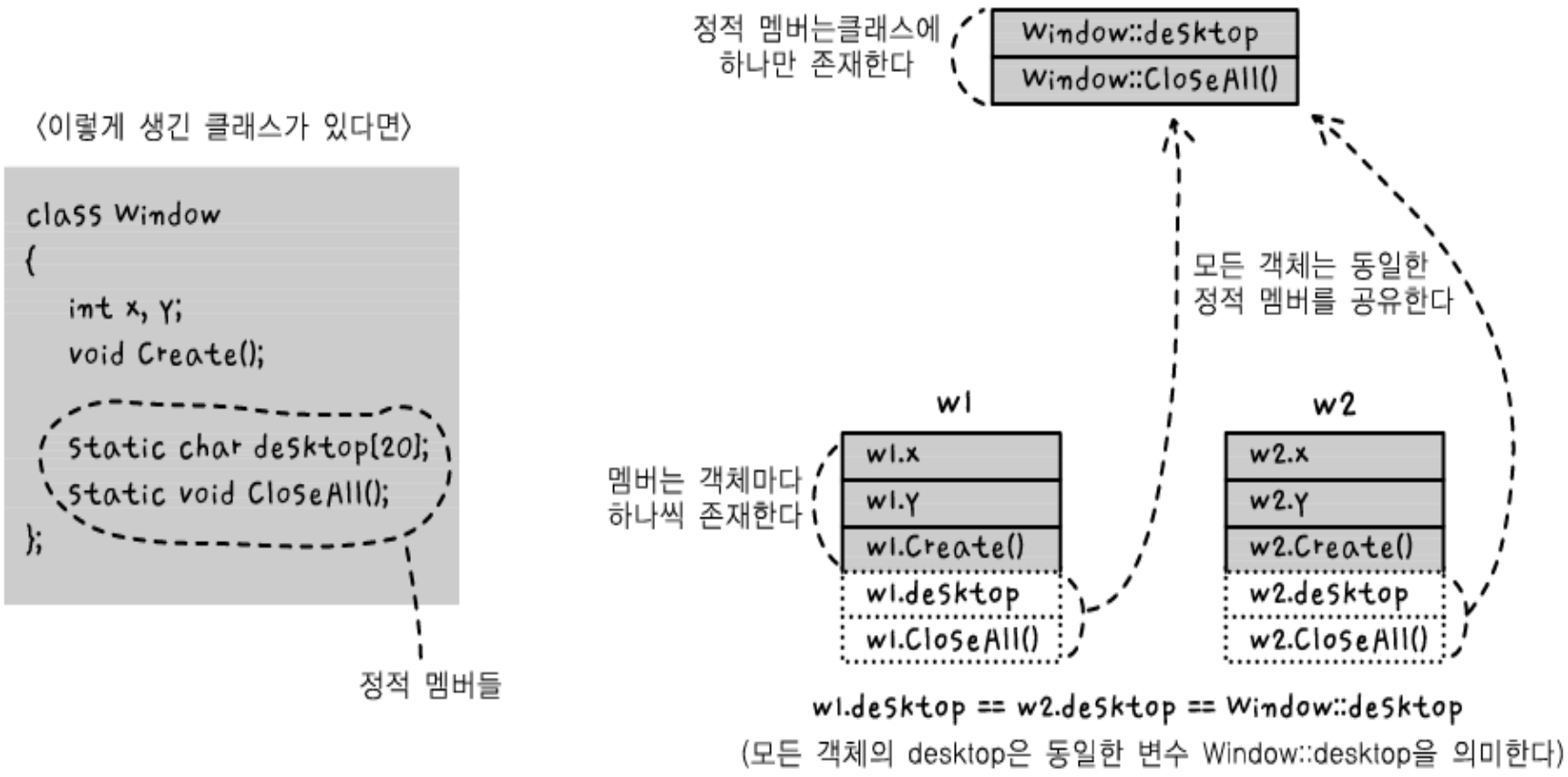
접근자

- 접근자란 외부에서 접근이 거부된 멤버 변수의 값을 읽거나, 변경하려는 용도의 멤버 함수를 말한다.

```
class Point
{
public:
          void SetX(int value)
          {
                    if (value < 0)
                              x = 0;
                    else if (value > 100)
                              x = 100;
                    else
                              x = value;
          }
          int GetX() {return x;};

// 중간 생략

private:
          // 멤버 변수
          int x, y;
};
```

정적 멤버 (Static Members)

- 정적 멤버란 모든 객체들이 공유하는 멤버를 말한다.



정적 멤버를 사용한 객체의 개수 세기(1)

- 정적 멤버를 사용해서 객체의 개수를 세는 예

```
class Student
{
public:
          string name;          // 이름
          int sNo;              // 학번
          Student(const string& name_arg, int stdNumber);
          ~Student();
public:
          // 정적 멤버들
          static int student_count;
          static void PrintStdCount();

};

// 정적 멤버 변수
int Student::student_count = 0;

// 정적 멤버 함수
void Student::PrintStdCount()
{
          cout << "Student 객체 수 = " <<  student_count << "\n";
}
```

정적 멤버를 사용한 객체의 개수 세기(2)

- 정적 멤버를 사용해서 객체의 개수를 세는 예

```
Student::Student(const string& name_arg, int stdNumber)
{
          // 학생 객체의 수를 증가시킨다.
          student_count++;

          name = name_arg;
          sNo = stdNumber;
}

Student::~Student()
{
          // 학생 객체의 수를 감소시킨다.
          student_count--;
}

void Func()
{
          Student std1("Bill", 342);
          Student std2("James", 214);

          Student::PrintStdCount();
}
```

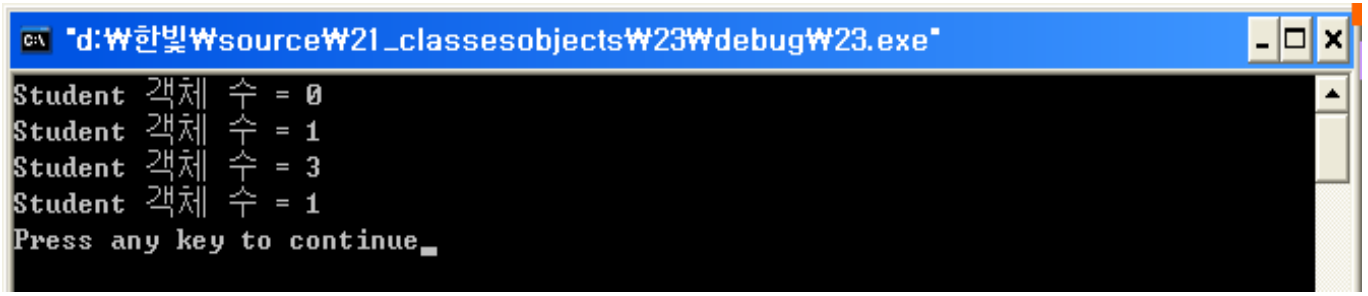
정적 멤버를 사용한 객체의 개수 세기(3)

- 정적 멤버를 사용해서 객체의 개수를 세는 예

```
int main()
{
        Student::PrintStdCount();

        Student std("Jeffrey", 123);

        Student::PrintStdCount();

        Func();

        Student::PrintStdCount();

        return 0;
}
```

[21-30]

- 실행 결과

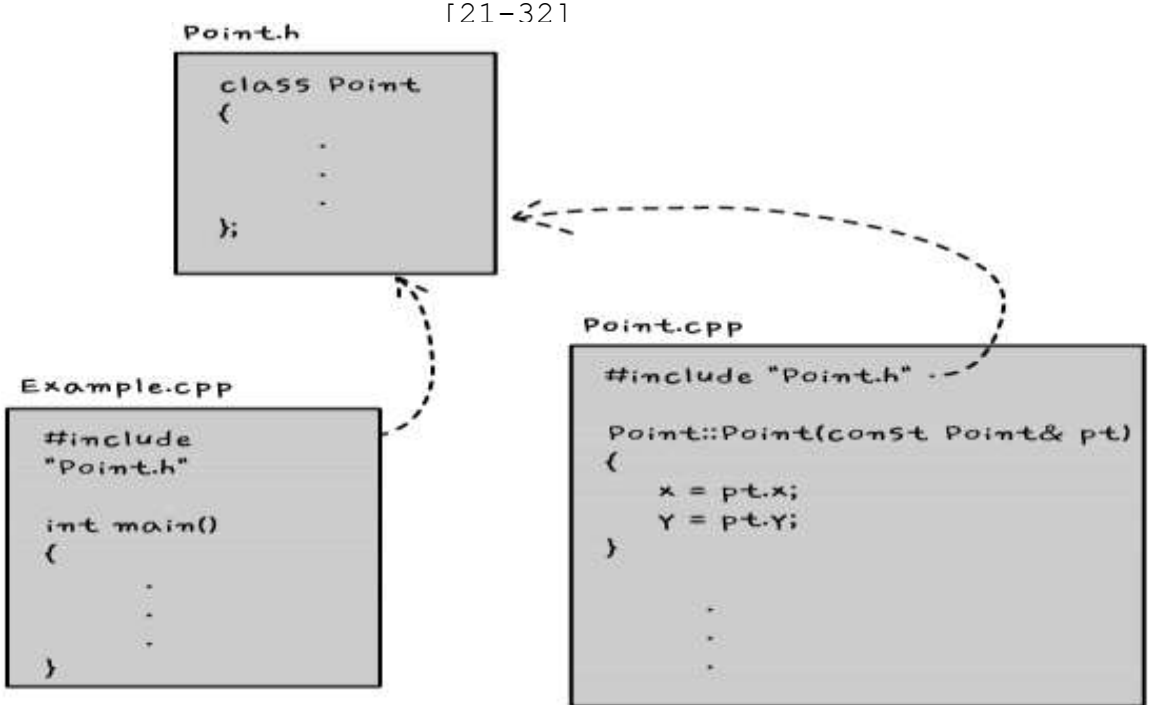


헤더 파일과 소스 파일로 나누기

- 헤더 파일에 넣어야 할 것과 구현 파일에 넣어야 할 것

헤더 파일	구현 파일
클래스의 정의 **inline 멤버 함수**	멤버 함수의 정의 정적 멤버 함수의 정의 정적 멤버 변수의 정의

- Point 클래스 예제를 헤더 파일과 구현 파일로 나눈 예



인라인 함수

- 함수를 인라인으로 만들면 실제로 함수가 호출되는 대신에, 함수를 호출한 위치에 함수의 내용이 그대로 옮겨진다.

```
// 인라인 함수
inline void Func()
{
          cout << "왜 인라인 함수를 쓰지?\n";
          cout << "도대체 인라인이 뭐야!!\n";
          cout << "뭐! 내가 인라인이야?\n";
}

int main()
{
          // 인라인 함수를 호출한다.
          Func();
          return 0;
}
```

||

```
int main()
{
          // 인라인 함수를 호출한다.
          cout << "왜 인라인 함수를 쓰지?\n";
          cout << "도대체 인라인이 뭐야!!\n";
          cout << "뭐! 내가 인라인이야?\n";
          return 0;
}
```

인라인 함수를 만드는 법

- 멤버 함수를 인라인으로 만드는 방법에는 두 가지가 있다.
  - 클래스의 내부에 정의한 멤버 함수들은 자동으로 인라인 함수가 된다.
  - 클래스의 외부에 정의한 멤버 함수는 함수의 정의 앞에 inline 키워드를 추가한다.

```
class CPoint
{
          // 중간 생략
};

inline void CPoint::SetX(int value)
{
          if (value < 0)                x = 0;
          else if (value > 100)         x = 100;
          else                          x = value;
}
inline void CPoint::SetY(int value)
{
          if (value < 0)                y = 0;
          else if (value > 100)         y = 100;
          else                          y = value;
}
```

인라인 함수의 사용 규칙

- 인라인 함수는 헤더 파일에 위치해야 한다.
  - 인라인 함수가 클래스의 정의 내부에 있는 경우
    - => 어차피 클래스의 정의는 헤더 파일에 위치하므로 특별히 신경 써줄 것이 없다.
  - 인라인 함수가 클래스의 정의 외부에 있는 경우
    - => 함수의 정의를 반드시 헤더 파일에 위치시켜야 한다.

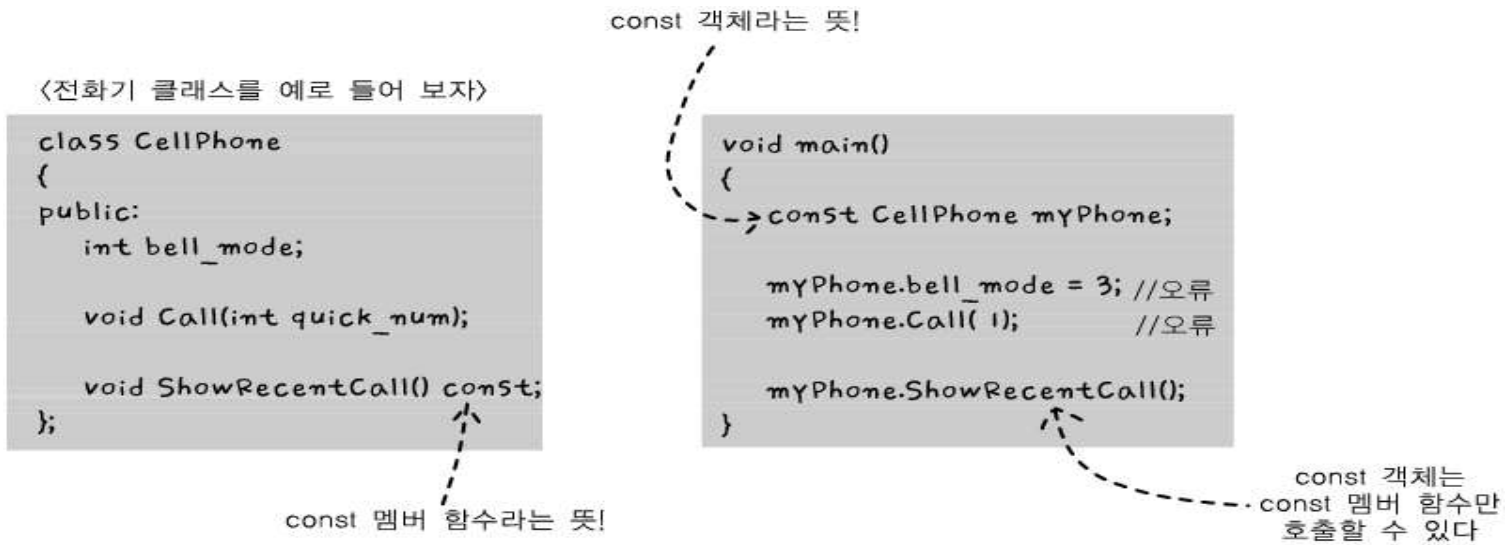
- 크기가 작은 함수만 인라인으로 만드는 것이 좋다.
  - 크기가 큰 함수를 인라인으로 만들면 실행 파일의 크기가 커지기 때문에 실행 성능이 낮아질 수 있다.

Const 함수

- 다음의 조건을 만족 하는 함수만 Const 함수로 만들 수 있다.
  - 멤버 변수의 값을 변경하지 않는 멤버 함수

- 멤버 함수를 Const 함수로 만들면 좋은 점
  - 다른 개발자가 "아, 이 함수는 멤버 변수의 값을 변경하지 않는구나" 라고 생각하게 만든다.
  - 실수로 멤버 변수의 값을 바꾸려고 하면, 컴파일러가 오류 메시지를 통해서 알려준다.
  - 객체가 Const 속성을 가진 경우에는 Const 함수만 호출할 수 있다.

[12-36]



Const 함수의 사용(1)

- 가능한 모든 함수를 Const 함수로 만드는 것이 좋다.

```
class Point
{
public:
          // 멤버 함수
          void Print() const;

          // 생성자들
          Point();
          Point(int initialX, int initialY);
          Point(const Point& pt);

          // 접근자
          void SetX(int value);
          void SetY(int value);
          int GetX() const {return x;}
          int GetY()       {return y;}  // 일부러 const 함수로 만들지 않음

private:
          // 멤버 변수
          int x, y;
};
```

멤버 함수에 대한 포인터(1)

- 멤버 함수에 대한 포인터 타입을 정의하는 예

```
#include "Point.h"

// void XX() 형태의 함수에 대한 포인터
typedef void (*FP1)(int);
// void Point::XX() 형태의 멤버 함수에 대한 포인터
typedef void (Point::*FP2)(int);

int main()
{
	// 객체를 생성한다.
	Point pt(50, 50);

	// FP1, FP2를 사용해서 Print() 함수를 가리킨다.
	// FP1 fp1 = &Point::SetX;	// 에러
	FP2 fp2 = &Point::SetX;		// 성공

	// 함수 포인터를 사용해서 함수 호출
	(pt.*fp2)(100);

	// 내용 출력
	pt.Print();

	return 0;
}
```

Const 함수의 사용(2)

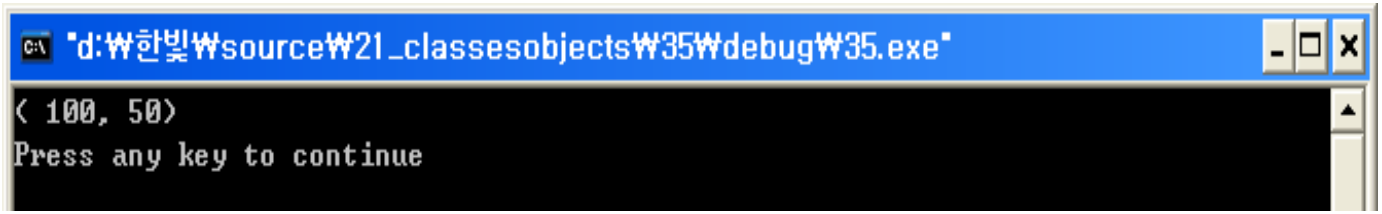
- Const 객체를 통해서 멤버 함수를 호출하는 예

```
void Area( const Point& pt)
{
          // (0,0)과 pt 가 이루는 사각형의 면적을 구한다.
          int area = pt.GetX() * pt.GetY();  // Error

          // 결과 출력
          cout << "(0, 0)과 이 점이 이루는 사각형의 면적 = " << area <<
"\n";
}
```

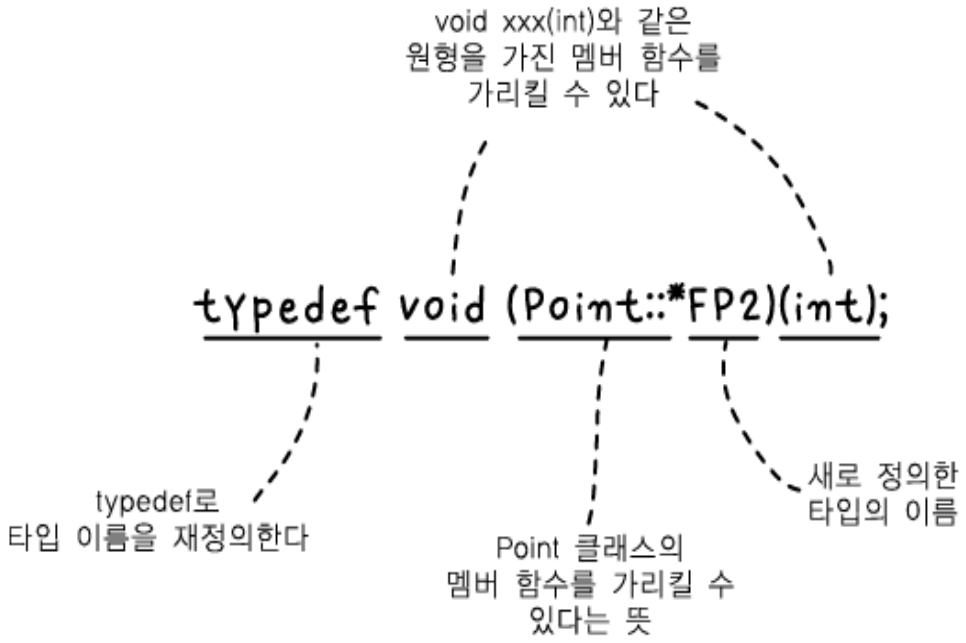
  - 매개변수 pt가 Const 객체이지만 CPoint::GetX()는 Const 함수이기 때문에 호출될 수 있다.
  - 그러나, CPoint::GetY() 는 Const 함수가 아니기 때문에 컴파일 오류를 발생시킨다.

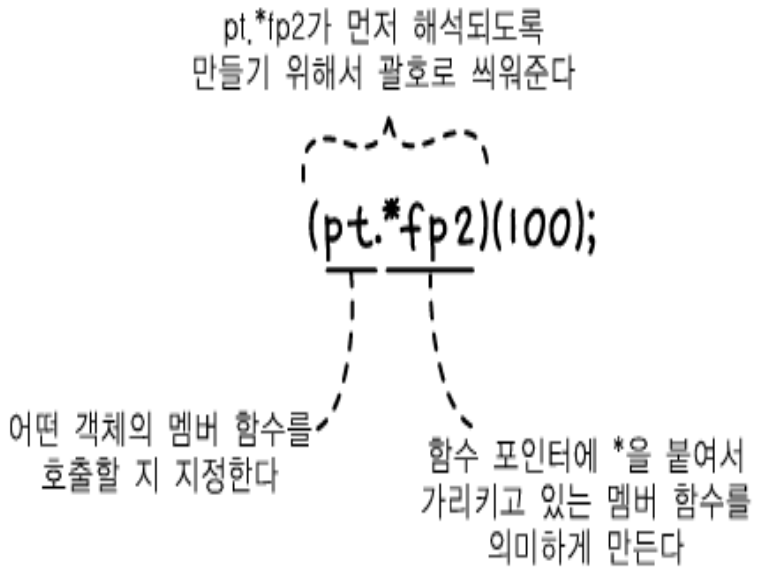
멤버 함수에 대한 포인터(2)

- 실행 결과



- 멤버 함수의 포인터 타입 정의와 호출





객체의 배열(1)

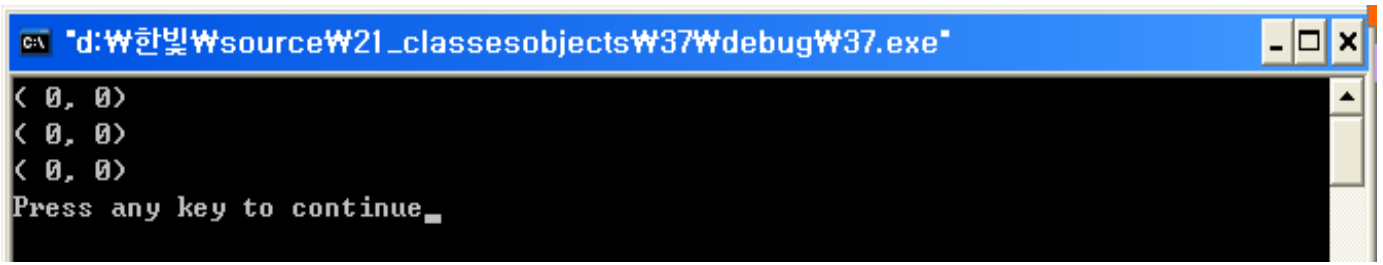
- 객체의 배열을 정의하는 경우 디폴트 생성자로 초기화 된다.

[21-38]

```
#include "Point.h"

int main()
{
          // 점 3개의 배열
          Point arr[3];

          // 모든 원소를 출력한다.
          for (int i = 0; i < 3; ++i)
                    arr[i].Print();

          return 0;
}
```

- 실행 결과

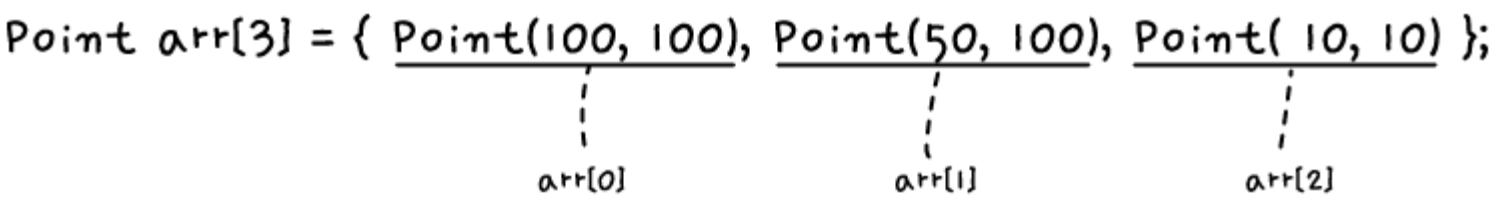
[21-42]



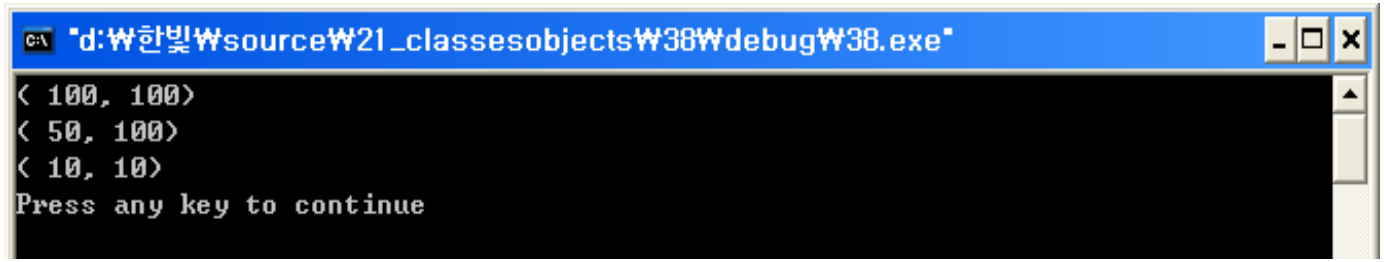
객체의 배열(2)

- 객체의 배열을 초기화하는 예

```
int main()
{
        // 점 3개의 배열
        Point arr[3] = { Point(100, 100), Point(50, 100), Point( 10, 10) };

        // 모든 원소를 출력한다.
        for (int i = 0; i < 3; ++i)
                arr[i].Print();

        return 0;
}
```



- 실행 결과

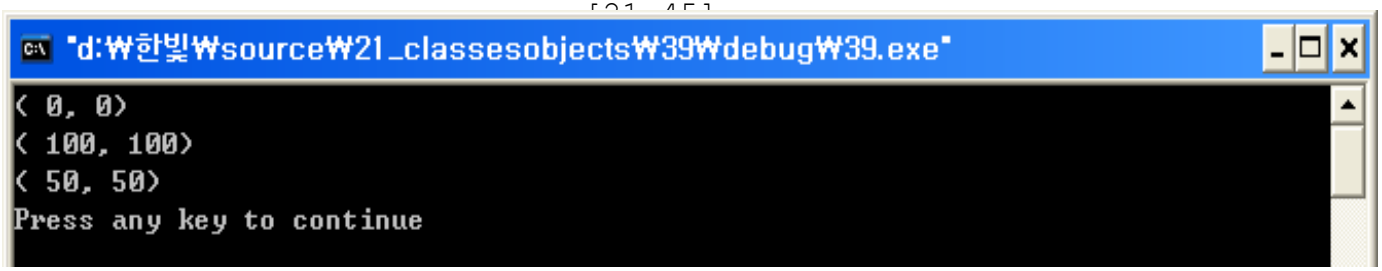


객체의 동적인 생성

- 동적 메모리 할당을 사용해서 객체를 생성하는 예

```
Point pt(50, 50);

Point* p1 = new Point();                    // 디폴트 생성자 사용
Point* p2 = new Point(100, 100);            // 인자있는 생성자 사용
Point* p3 = new Point( pt);                 // 복사 생성자 사용

p1->Print();
p2->Print();
p3->Print();

delete p1;
delete p2;
delete p3;
```

- 실행 결과



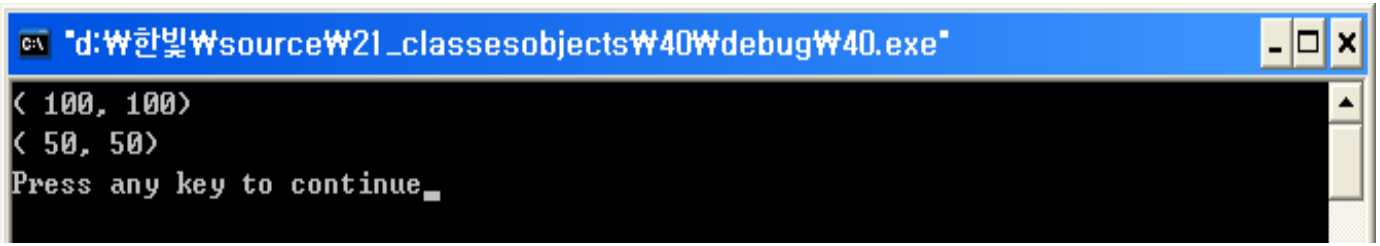
생성자와 소멸자의 호출 시점(1)

- 동적 메모리 할당을 사용한 경우의 생성자와 소멸자의 호출 시점을 알아보자.

```
int main
{
          Point pt(100, 100);

          Point* p = 0;

          p = new Point(50, 50);

          pt.Print();
          p->Print();

          // 동적으로 생성한 객체를 해제한다.
          delete p;
          p = NULL;

          return 0;
}
```

생성자와 소멸자의 호출 시점(2)

- 실행 결과



- 생성자와 소멸자의 호출 시점을 표로 정리

	정적인 생성	동적인 생성
생성자의 호출	객체를 정의할 때	new 연산자로 할당할 때
소멸자의 호출	객체를 정의한 함수가 끝날 때	delete 연산자로 해제할 때

클래스 안에 넣을 수 있는 다른 것들

- 열거체나 typedef을 클래스 안에서 정의하는 예

```
class Point
{
public:
          enum { MIN_X = 0, MAX_X = 100, MIN_Y = 0, MAX_Y = 100 };
          typedef int COOR_T; // 좌표의 타입

          // 멤버 함수
          void Print() const;
          void Offset(COOR_T x_delta, COOR_T y_delta);
          void Offset(const Point& pt);

          // 생성자들
          Point();
          Point(COOR_T initialX, COOR_T initialY);
          Point(const Point& pt);

          // 접근자
          void SetX(COOR_T value);
          void SetY(COOR_T value);
          COOR_T GetX() const {return x;};
          COOR_T GetY() const {return y;};

private:
          // 멤버 변수
          COOR_T x, y;
};
```

This 포인터(1)

- 멤버 함수 안에서 자기 자신의 주소를 확인하는 예

```
class WhoAmI
{
public:
          int id;

          WhoAmI(int id_arg);
          void ShowYourself() const;
};

WhoAmI:: WhoAmI(int id_arg)
{
          id = id_arg;
}

void WhoAmI::ShowYourself() const
{
          cout << "{ID = " << id << ", this = " << this << "}\n";
}
```

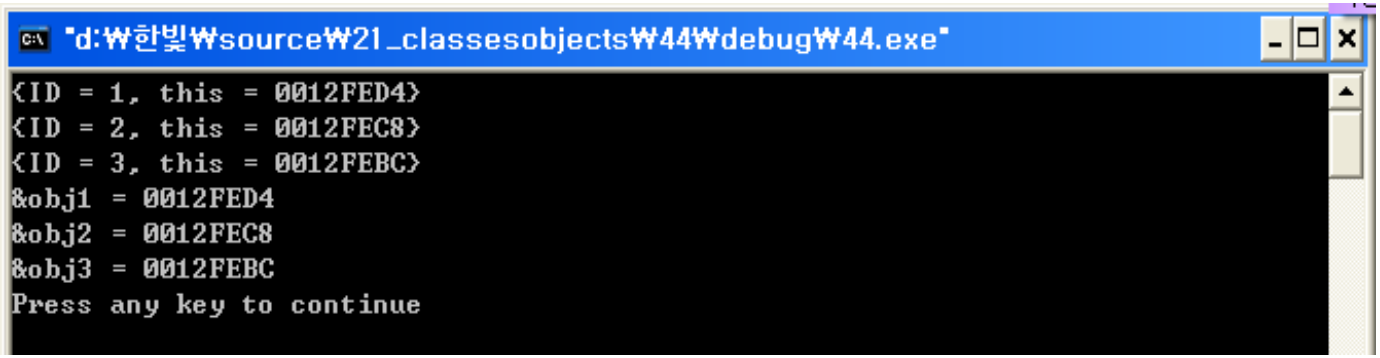
This 포인터(2)

- 멤버 함수 안에서 자기 자신의 주소를 확인하는 예

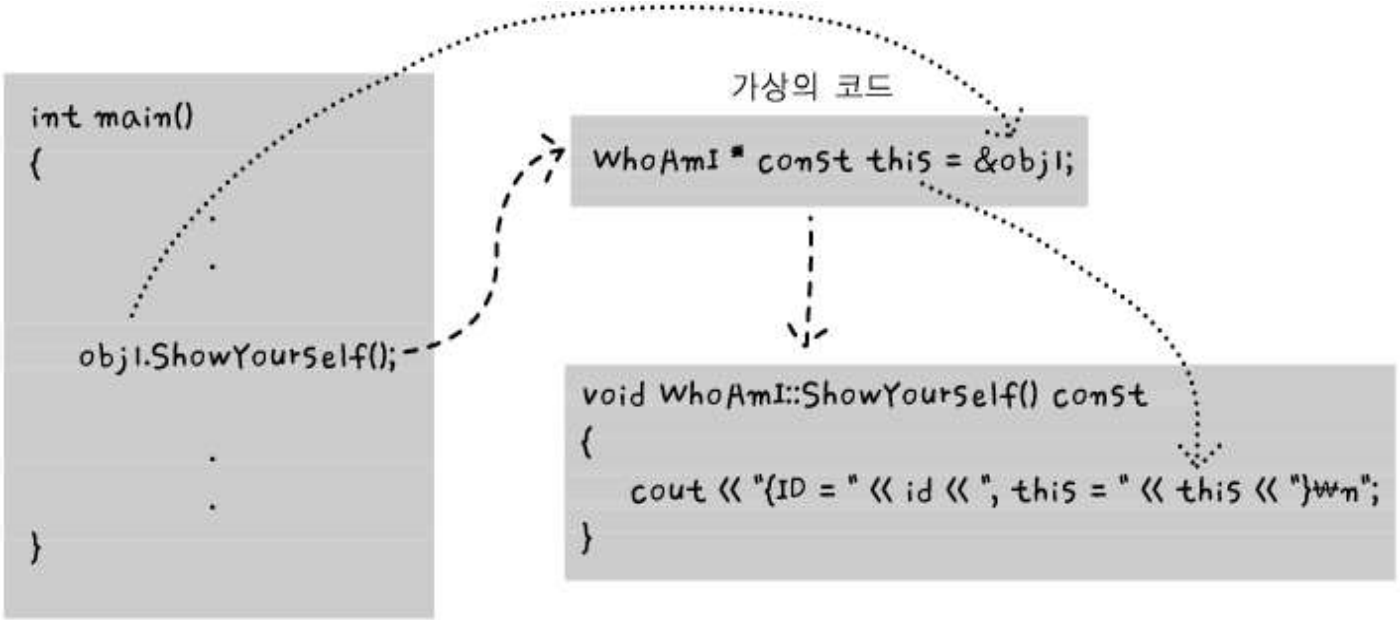
```
int main()
{
          // 세 개의 객체를 만든다.
          WhoAmI obj1( 1);
          WhoAmI obj2( 2);
          WhoAmI obj3( 3);

          // 객체들의 정보를 출력한다.
          obj1.ShowYourself();
          obj2.ShowYourself();
          obj3.ShowYourself();

          // 객체들의 주소를 출력한다.
          cout << "&obj1 = " << &obj1 << "\n";
          cout << "&obj2 = " << &obj2 << "\n";
          cout << "&obj3 = " << &obj3 << "\n";

          return 0;
}
```

This 포인터(3)

- 실행 결과



- 멤버 함수의 호출과 가상의 코드



Q&A