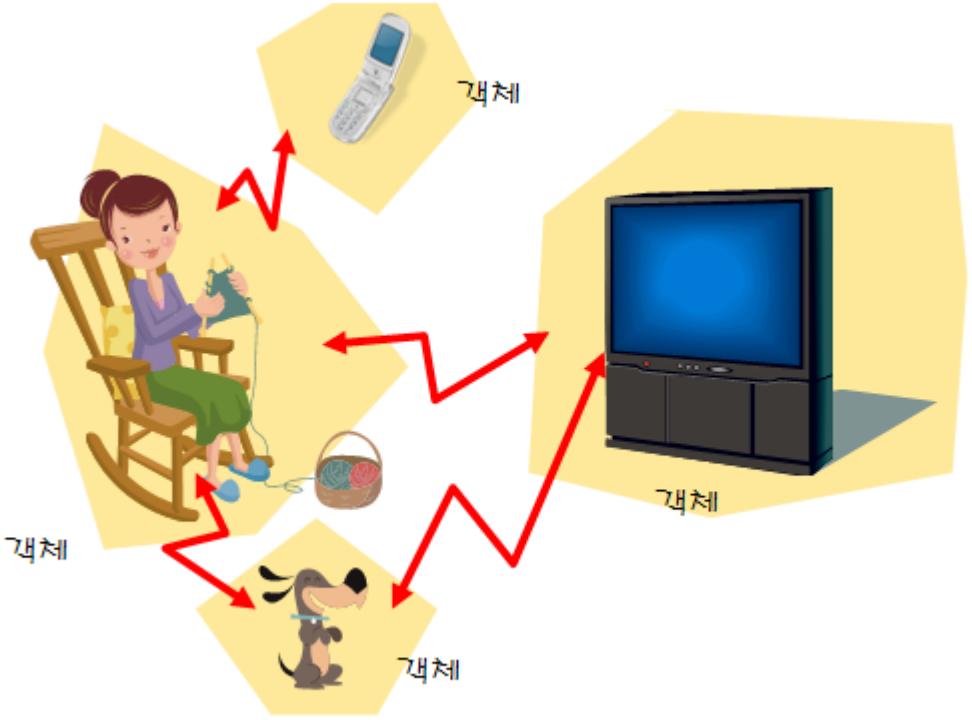
C++ Espresso

수업 보충

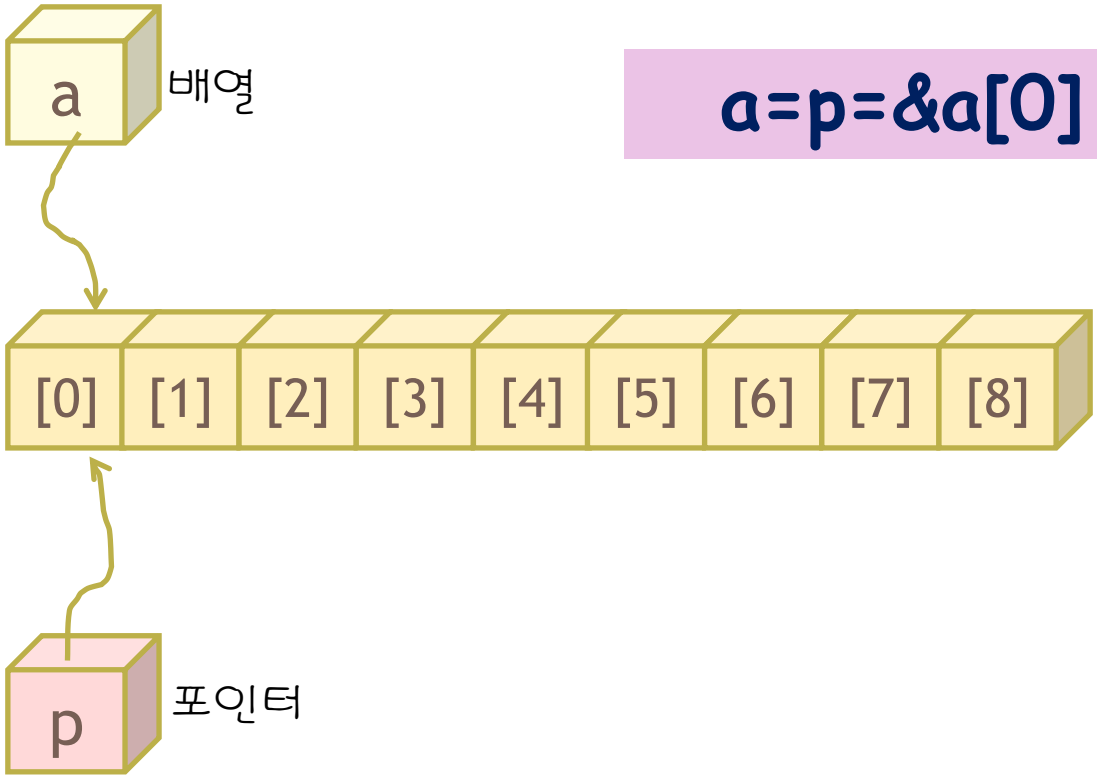


동적 메모리 할당 및 반환

- 정적 할당
  - 변수 선언을 통해 필요한 메모리 할당
    - 많은 양의 메모리는 배열 선언을 통해 할당
- 동적 할당
  - 필요한 양이 예측되지 않는 경우. 프로그램 작성시 할당 받을 수 없음
  - 실행 중에 운영체제로부터 할당 받음
    - 힙(heap)으로부터 할당
      - 힙은 운영체제가 소유하고 관리하는 메모리. 모든 프로세스가 공유할 수 있는 메모리

포인터와 배열

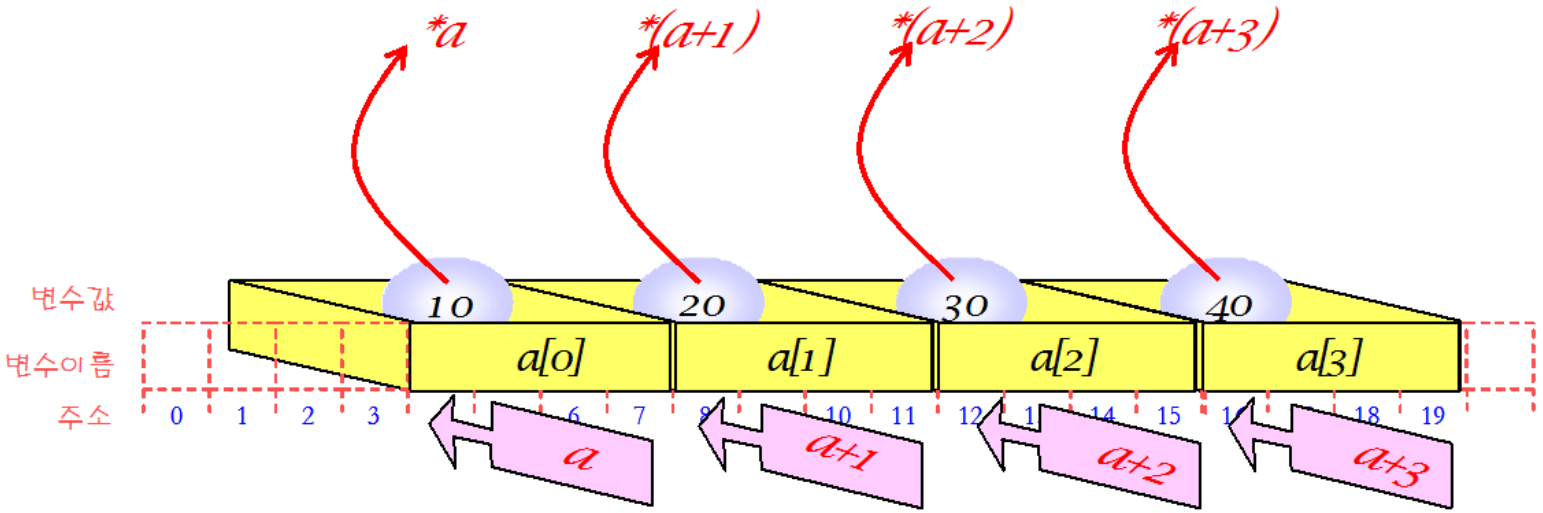
- 배열과 포인터는 아주 밀접한 관계를 가지고 있다.
- 배열 이름이 바로 포인터이다.
- 포인터는 배열처럼 사용이 가능하다



a=p=&a[0]

배열과 포인터의 관계

인덱스 표기법	포인터 표기법	설명
a[0]	*a	배열의 첫번째 요소값
a[1]	*(a+1)	배열의 두번째 요소값
a[2]	*(a+2)	배열의 세번째 요소값
…	…	…



포인터를 사용한 원소의 탐색

- 일반적인 방법을 사용한 원소의 탐색

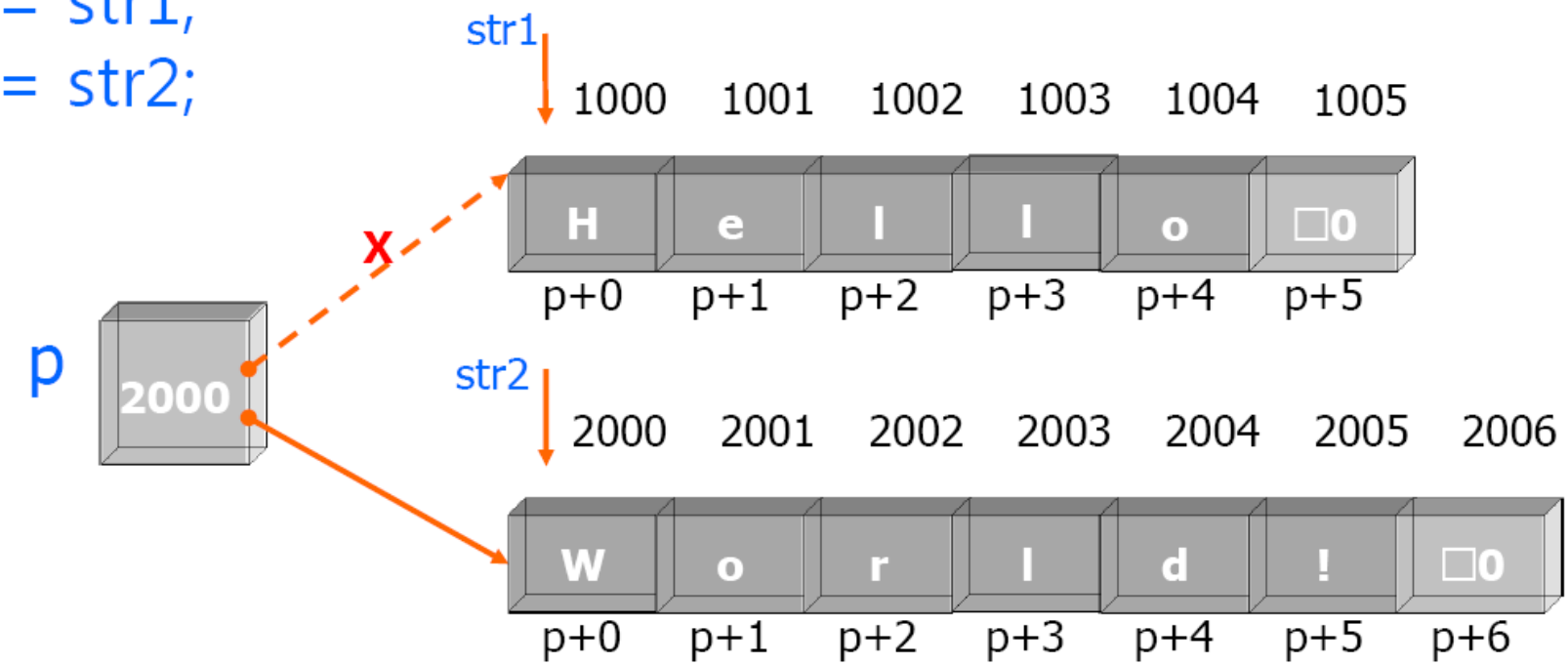
```
int nArray[10];

 for (int i = 0; i < 10; ++i) // 배열을 탐색하면서 값을 넣는다.
    nArray[i] = i;
```

- 포인터를 사용한 원소의 탐색

```
int nArray[10];
int* p = &nArray[0];

for (int i = 0; i < 10; ++i) // 배열을 탐색하면서 0으로 초기화
    [redacted]
```

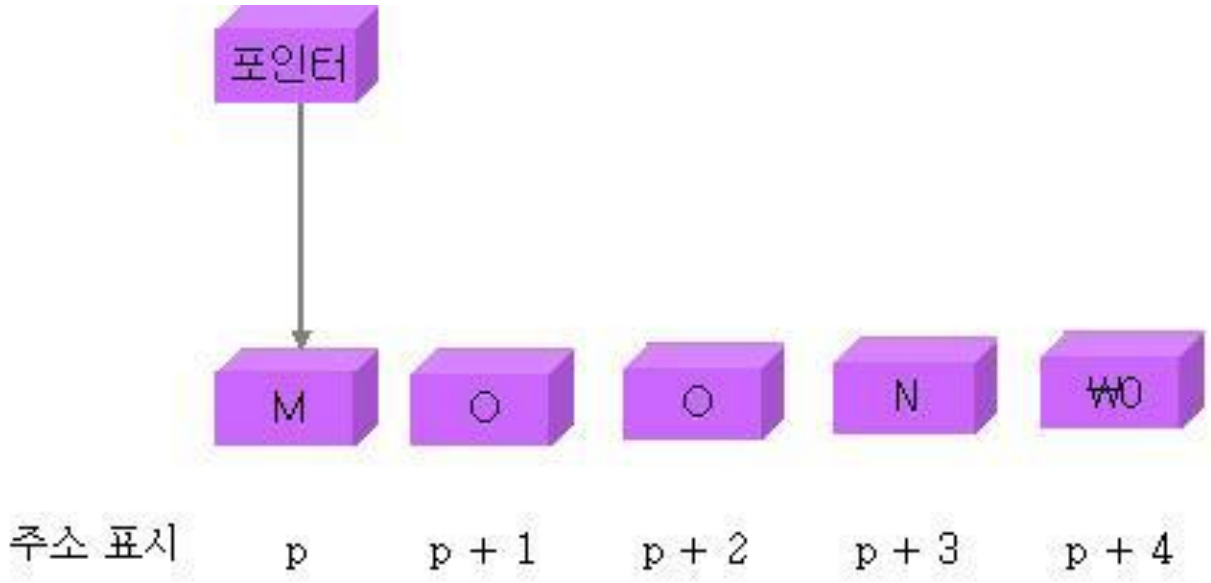
문자열과 포인터

```
char str1[] = "Hello";
char str2[] = "World!";
char *p;
p = str1;
p = str2;
```



포인터를 이용한 문자열 표현

- char *포인터 이름 = “ 문자열 ” ;
  - char *p = "MOON";

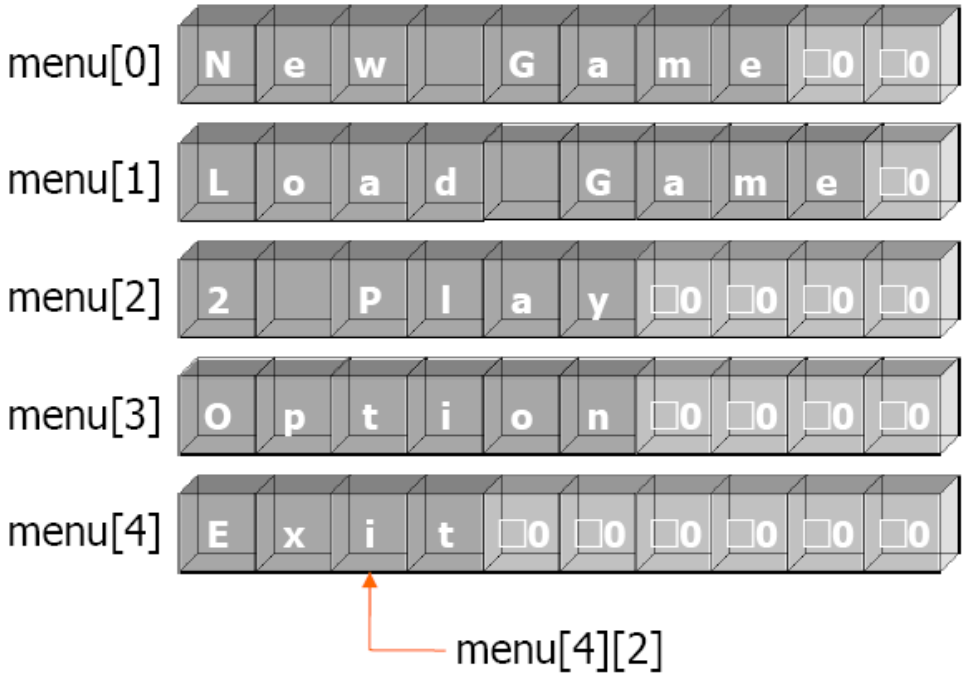


문자열 배열

각 문자열의 길이가 최대 10자인 문자열 배열 5개를 선언

```
char menu[5][10] = { "New Game", "Load Game", "2 Play", "Option", "Exit" };
```

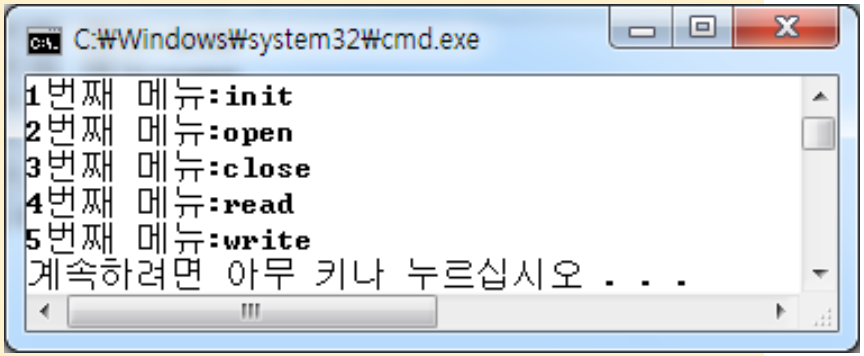
생략가능



#실습
(Practice1.cpp)

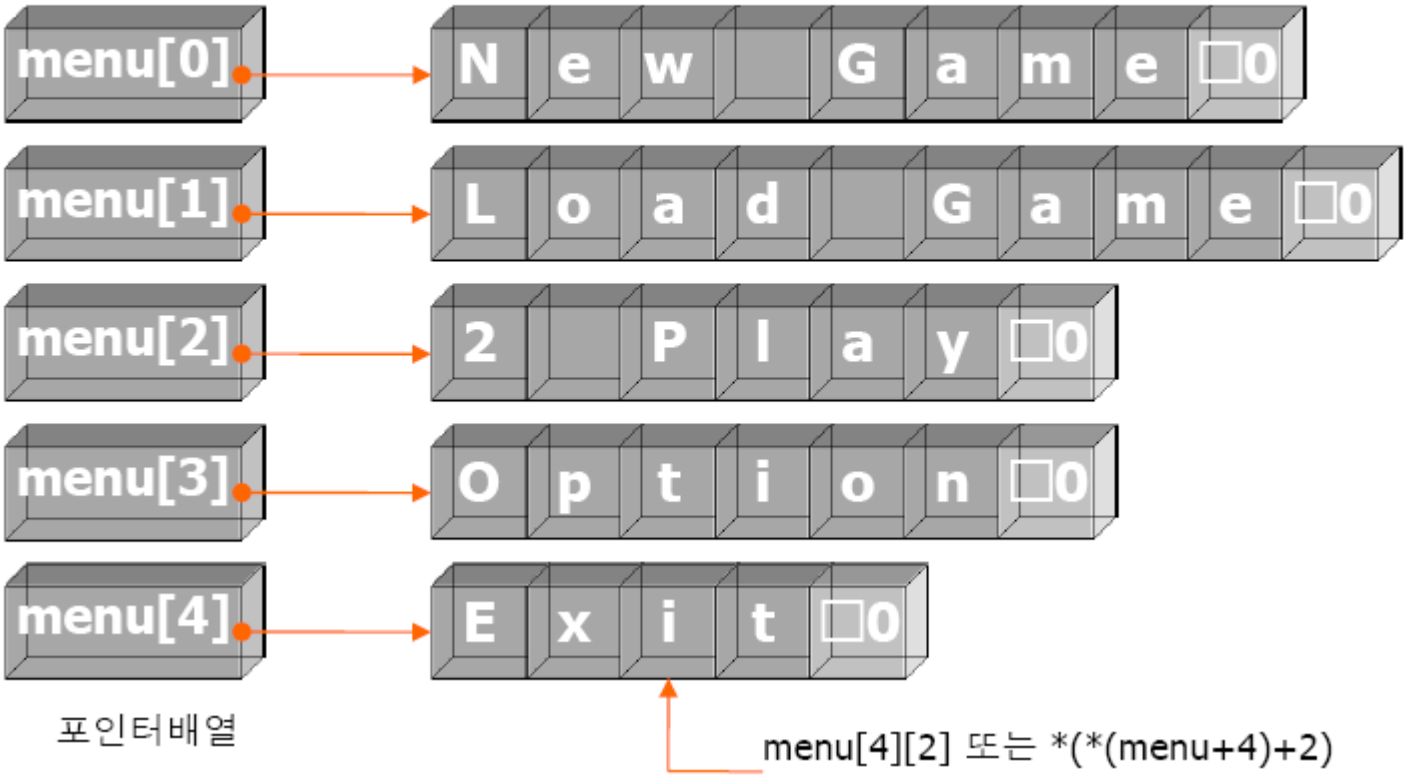
예제

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

int main(void)
{
        int i;
        char menu[5][10] = {
                        "init",
                        "open",
                        "close",
                        “read",
                        "write"
        };

for (i = 0; i < 5; i++)
cout <<i<<"번째 메뉴:"<< menu[i]<<endl;
return 0;
}
```



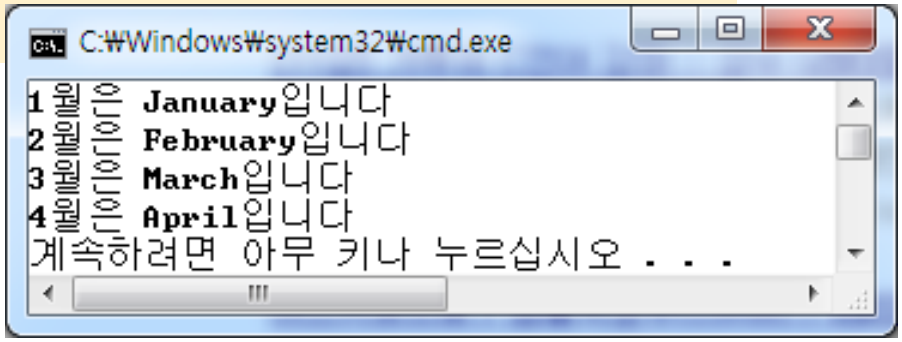
문자열 배열

char *menu[] = { "New Game", "Load Game", "2 Play", "Option", "Exit" };



#실습 (Practice1.cpp)

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

int main(void)
{
	char *p[4] = { "January", "February", "March","April" };
	int i;

	for (i = 0; i < 4; i++)
		cout<<i+1<<"월은 "<< p[i]<<"입니다"<<endl;
	return 0;
}
```



this 포인터

- this
  - 포인터, 객체 자신 포인터
  - 클래스의 멤버 함수 내에서만 사용
  - 개발자가 선언하는 변수가 아니고, 컴파일러가 선언한 변수
    - 멤버 함수에 컴파일러에 의해 묵시적으로 삽입 선언되는 매개 변수

```
class Circle {
   int radius;
public:
   Circle() { this->radius=1; }
   Circle(int radius) { this->radius = radius; }
   void setRadius(int radius) { this->radius = radius; }
   ....
};
```

this와 객체

* 각 객체 속의 this는 다른 객체의 this와 다름

this는 객체 자신에 대한 포인터

c1
radius 4
…
void setRadius(int radius) {
this->radius = radius;
}
…

c2
radius 5
…
void setRadius(int radius) {
this->radius = radius;
}
…

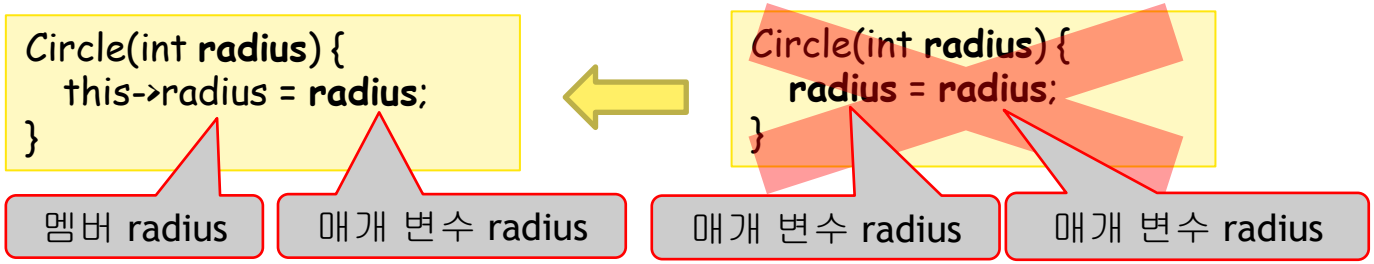
c3
radius 6
…
void setRadius(int radius) {
this->radius = radius;
}
…

```
class Circle {
   int radius;
public:
   Circle() {
      this->radius=1;
   }
   Circle(int radius) {
      this->radius = radius;
   }
   void setRadius(int radius) {
      this->radius = radius;
   }
};
```

```
int main() {
   Circle c1;
   Circle c2(2);
   Circle c3(3);

   c1.setRadius(4);
   c2.setRadius(5);
   c3.setRadius(6);
}
```

this가 필요한 경우

- 매개변수의 이름과 멤버 변수의 이름이 같은 경우



- 멤버 함수가 객체 자신의 주소를 리턴할 때
  - 연산자 중복 시에 매우 필요

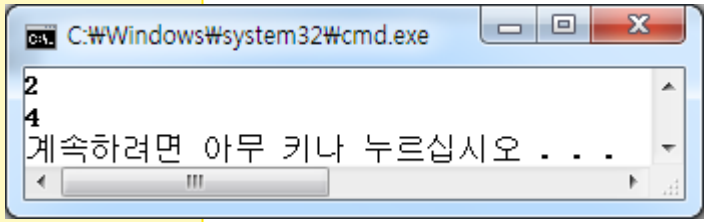
```
class Sample {
public:
   Sample* f() {
      ....
      return this;
   }
};
```

this의 제약 사항

- 멤버 함수가 아닌 함수에서 this 사용 불가
  - 객체와의 관련성이 없기 때문
- static 멤버 함수에서 this 사용 불가
  - 객체가 생기기 전에 static 함수 호출이 있을 수 있기 때문에

출력 결과를 예상해 보세요~~

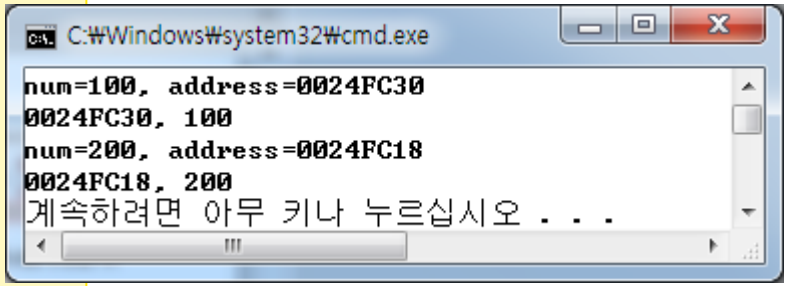
```
#include <iostream>
using namespace std;

class TwoNumber{
private:
        int num1;
        int num2;
public:
        TwoNumber(int num1, int num2){
                this->num1=num1;
                this->num2=num2;
        }

        void ShowTwoNumber(){
                cout<<this->num1<<endl;
                cout<<this->num2<<endl;
        }
};

int main(void){
        TwoNumber two(2, 4);
        two.ShowTwoNumber();
        return 0;
}
```



#실습
(Practice1.cpp)

예제

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

class SoSimple{
        int num;
public:
        SoSimple(int n) : num(n){
                cout<<"num="<<num<<", ";
                cout<<"address="<<this<<endl;
        }

        void ShowSimpleData(){
                cout<<num<<endl;
        }

        SoSimple * GetThisPointer(){
                return this;
        }
};
```




```
int main(void){
        SoSimple sim1(100);
        SoSimple * ptr1=sim1.GetThisPointer();
        cout<<ptr1<<", ";
        ptr1->ShowSimpleData();

        SoSimple sim2(200);
        SoSimple * ptr2=sim2.GetThisPointer();
        cout<<ptr2<<", ";
        ptr2->ShowSimpleData();
        return 0;
}
```

this 포인터의 실체 – 컴파일러에서 처리

```
class Sample {
   int a;
public:
   void setA(int x) {
      this->a = x;
   }
};
```

컴파일러에 의해 변환

```
class Sample {
   ....
public:
   void setA(Sample* this, int x) {
      this->a = x;
   }
};
```

this는 컴파일러에 의해 묵시적으로 삽입된 매개 변수

(a) 개발자가 작성한 클래스

(b) 컴파일러에 의해 변환된 클래스

```
Sample ob;
ob.setA(5);
```

컴파일러에 의해 변환

```
ob.setA(&ob, 5);
```

ob의 주소가 this 매개변수에 전달됨

(c) 객체의 멤버 함수를 호출하는 코드의 변환

Overloading & overriding

- 오버로딩(Overloading) - 함수 중복 정의

: 이름의 함수에 매개변수를 다르게 사용하여 매개 변수에 따라 다른 함수가 실행되는 것.

- 오버라이딩(Overriding) - 함수 재정의

: 부모 클래스의 멤버 함수를 자식 클래스에서 다시 정의하는 것