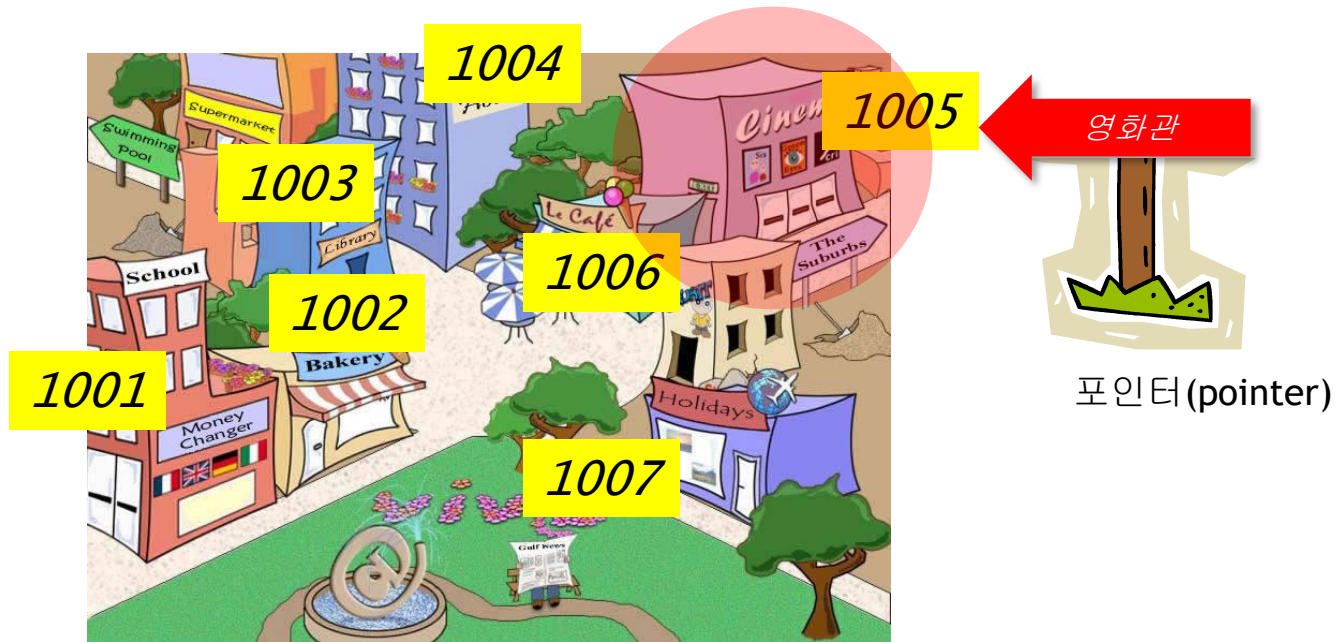


포인터 (pointer)

포인터란?

- *포인터(pointer)*: 주소를 가지고 있는 변수

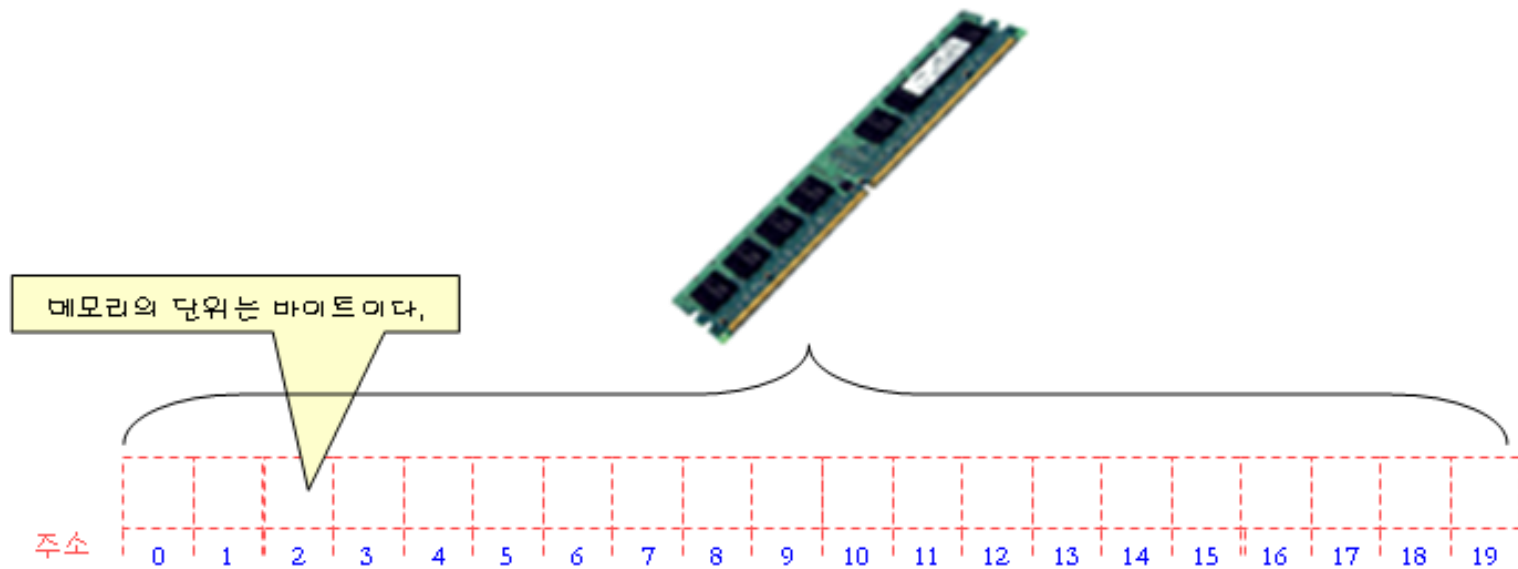






메모리의 구조

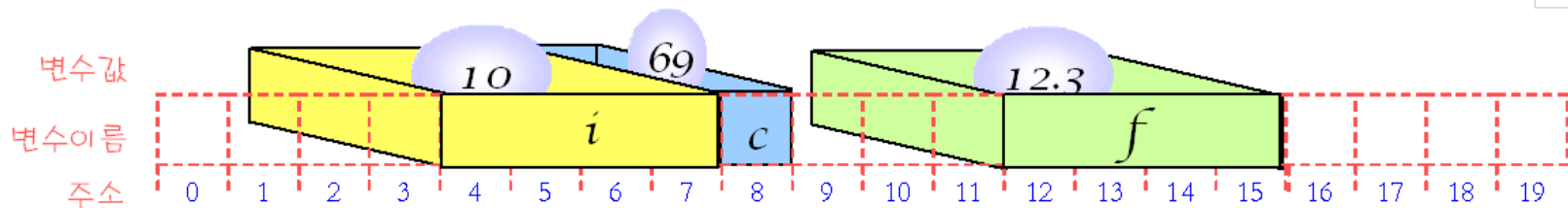
- 변수는 메모리에 저장된다.
- 메모리는 바이트 단위로 액세스된다.
 - 첫번째 바이트의 주소는 0, 두번째 바이트는 1,...



변수와 메모리

- 변수의 크기에 따라서 차지하는 메모리 공간이 달라진다.
- char형 변수: 1바이트, int형 변수: 4바이트,...

```
int main()
{
    int i = 10;
    char c = 69;
    float f = 12.3;
}
```

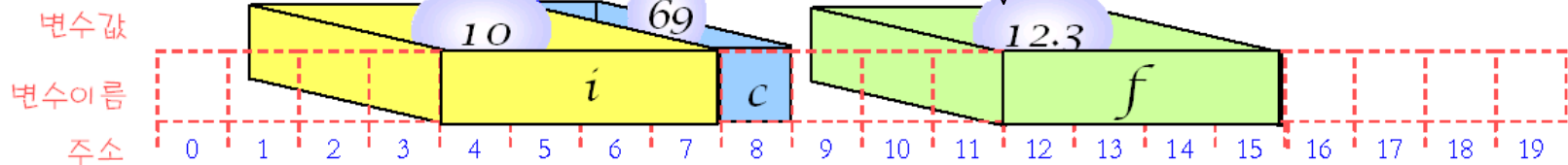


변수와 메모리

- 변수의 메모리상 배치되는 위치는 ?
 - 사용하고 있는 컴퓨터나 실행환경에 따라 달라짐
 - 변수의 실제 주소값이 얼마인지는 중요하지 않음

```
int main()
{
    int i = 10;
    char c = 69;
    float f = 12.3;
}
```

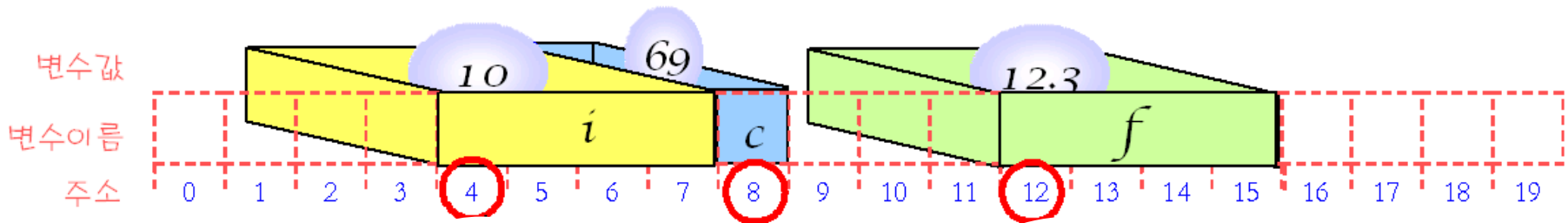
변수의 저장 위치
알 수 있음



변수의 주소

- 변수의 주소를 계산하는 연산자: **&**

```
int main()
{
    int i = 10;
    char c = 69;
    float f = 12.3;
}
```



변수의 주소

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

int main()
{
    int i = 10;
    char c = 69;
    float f = 12.3;

    cout << "i의 주소 : " << hex << &i << endl;    // 변수 i의 주소 출력
    cout << "c의 주소 : " << hex << &c << endl;    // 변수 c의 주소 출력
    cout << "f의 주소 : " << hex << &f << endl;    // 변수 f의 주소 출력
}
```

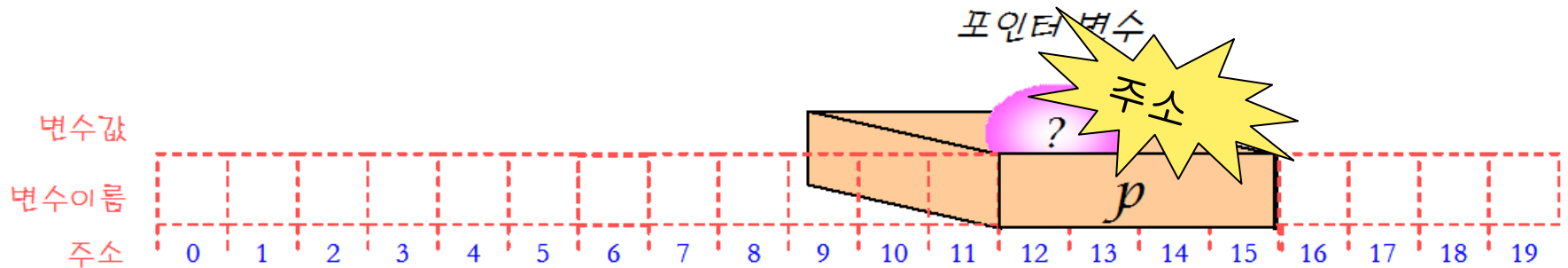


i의 주소: 1245024
c의 주소: 1245015
f의 주소: 1245000

포인터의 선언

- **포인터**: 변수의 주소를 가지고 있는 변수

```
int * p;
```

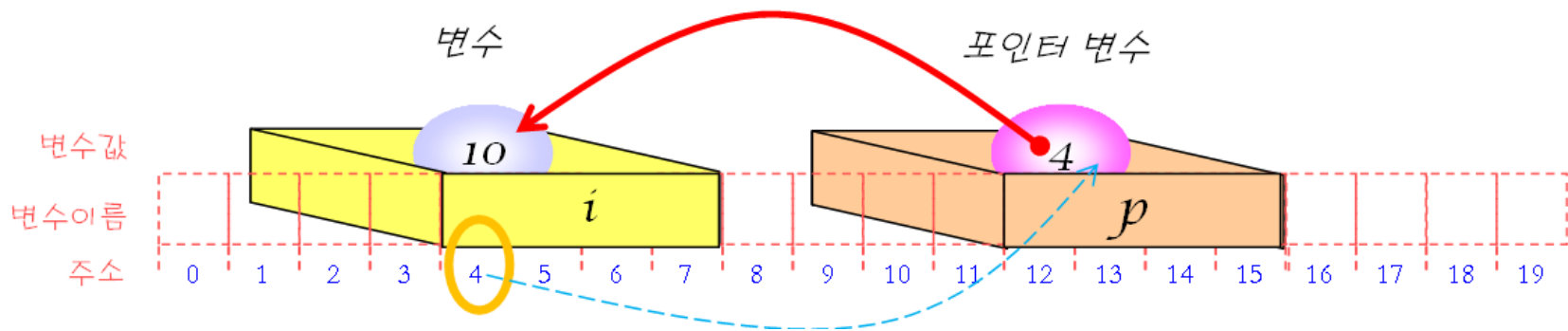


포인터와 변수의 연결

```
int main()
{
    int i = 10;           // 정수형 변수 i 선언
    int* p;               // 포인터 변수 p 선언

    p = &i;               // 변수 i의 주소가 포인터 p로 대입

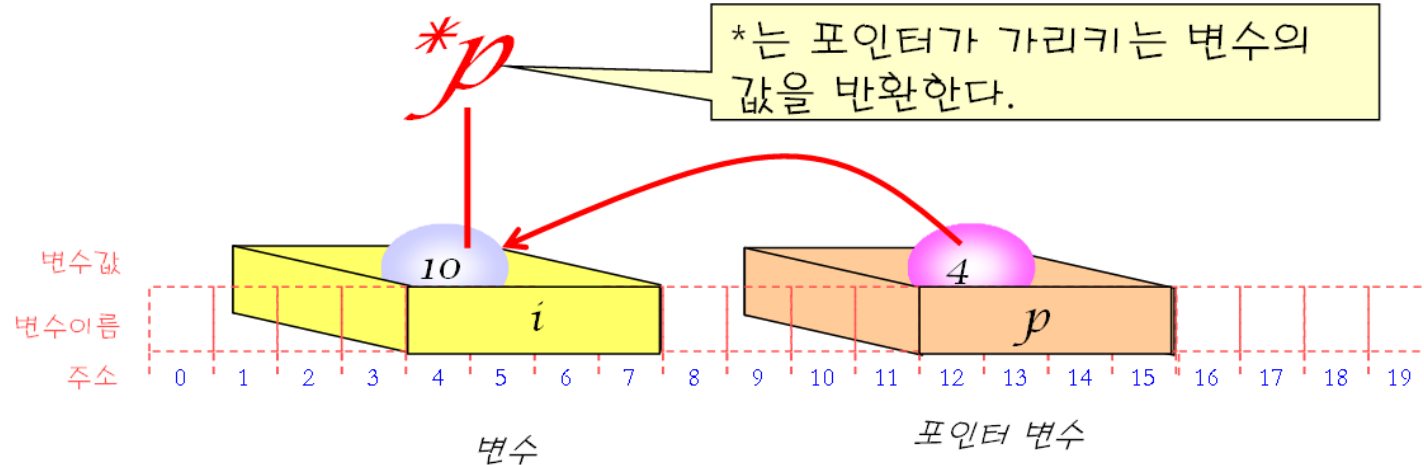
    cout << "i의 주소 : " << hex << &i << endl;
    cout << "p의 내용 : " << hex << p << endl;
}
```



간접 참조 연산자

- 간접 참조 연산자 ***** : 포인터가 가리키는 값을 가져오는 연산자

```
int i=10;  
int *p;  
p = &i;  
printf("%d", *p):
```



포인터와 변수 연결

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

```
int main()
{
```

```
    int i = 10;
```

```
    int* p;
```

```
    p = &i;
```

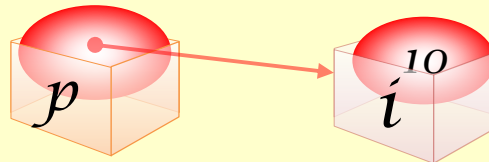
```
    cout << "i의 값 : " << hex << i << endl;    // 변수 i의 주소 출력
```

```
    cout << "i의 주소 : " << hex << &i << endl;    // 변수 i의 주소 출력
```

```
    cout << "p의 값 : " << hex << p << endl;    // 포인터 p의 내용 출력
```

```
    cout << "*p의 내용 : " << hex << *p << endl;    // 포인터 p의 내용 출력
```

```
}
```



i의 값=10

i의 주소=0014FE60

p의 값=0014FE60

*p의 내용=10

포인터의 값은 변경가능

- 포인터 값(주소)은 변경할 수 있다!
 - 실행 도중에 포인터가 가리키는 변수를 변경할 수 있음

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

```
int main()
{
    int x = 100;
    int y = 200;
```

```
    int* p;
```

```
    p = &x;
```

```
    cout << "p의 내용 : " << hex << p << endl;    // 포인터 p의 내용 출력
```

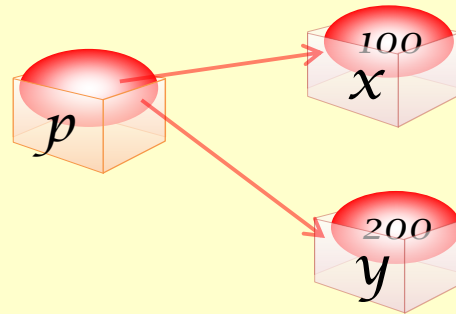
```
    cout << "*p의 내용 : " << dec << *p << endl;    // 포인터 p의 내용 출력
```

```
    p = &y;
```

```
    cout << "p의 내용 : " << hex << p << endl;    // 포인터 p의 내용 출력
```

```
    cout << "*p의 내용 : " << dec << *p << endl;    // 포인터 p의 내용 출력
```

```
}
```



p의 값=3144220
*p의 값=100
p의 값=3144208
*p의 값=200

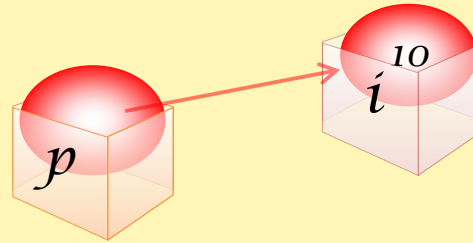
포인터를 통하여 값 변경

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

```
int main()
{
    int i = 10;
    int* p;

    p = &i;
    cout << i << endl;

    *p = 20;
    cout << i << endl;    // ?
}
```



포인터를 통하여 변수의 값을 변경한다.

```
i = 10
i = 20
```

포인터 사용시 주의점

- 초기화 하지 않은 포인터를 사용하면 안된다.

```
int main()
{
    int *p;           // 포인터 p는 초기화가 안 되어 있음
    *p = 100;         // 위험한 코드
}
```



포인터 사용시 주의점

- 포인터의 타입과 변수의 타입은 일치하여야 한다.

```
int main(void)
{
    int i;
    double *pd;

    pd = &i;
    *pd = 36.5;

    return 0;
}
```

// 오류! double형 포인터에 int형 변수의 주소를 대입

포인터 연산

- 가능한 연산: 증가, 감소, 덧셈, 뺄셈 연산
- 증가 연산의 경우 증가되는 값은 포인터가 가리키는 객체의 크기

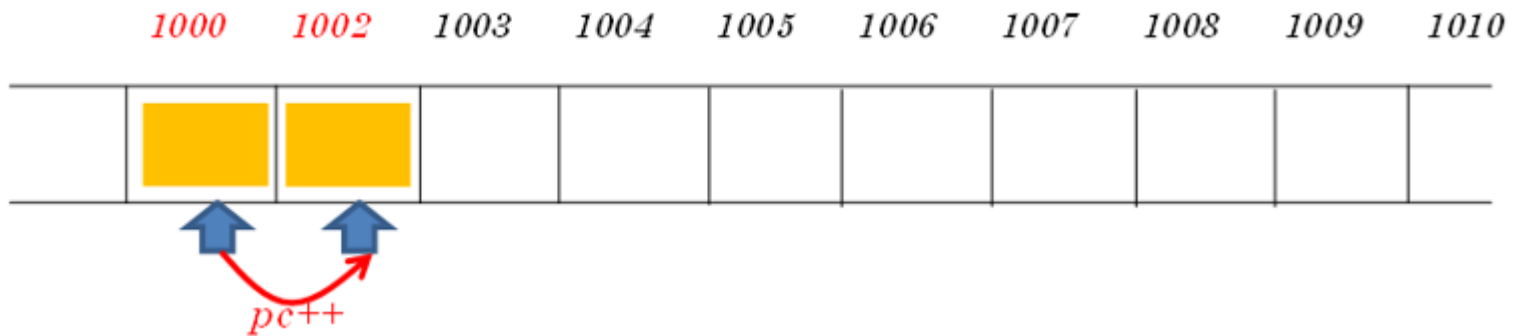
p++;

포인터 타입	++연산 후 증가되는값
char	1
short	2
int	4
float	4
double	8

포인터의 증가
→ 가리키는 객체의
크기만큼 증가

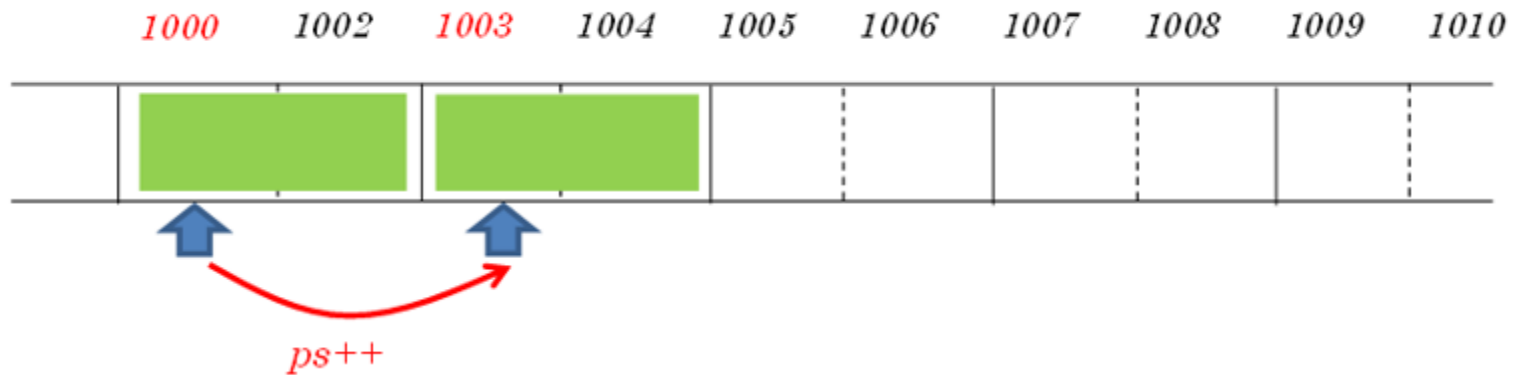
포인터의 증감 연산

```
char *pc;  
pc++;
```



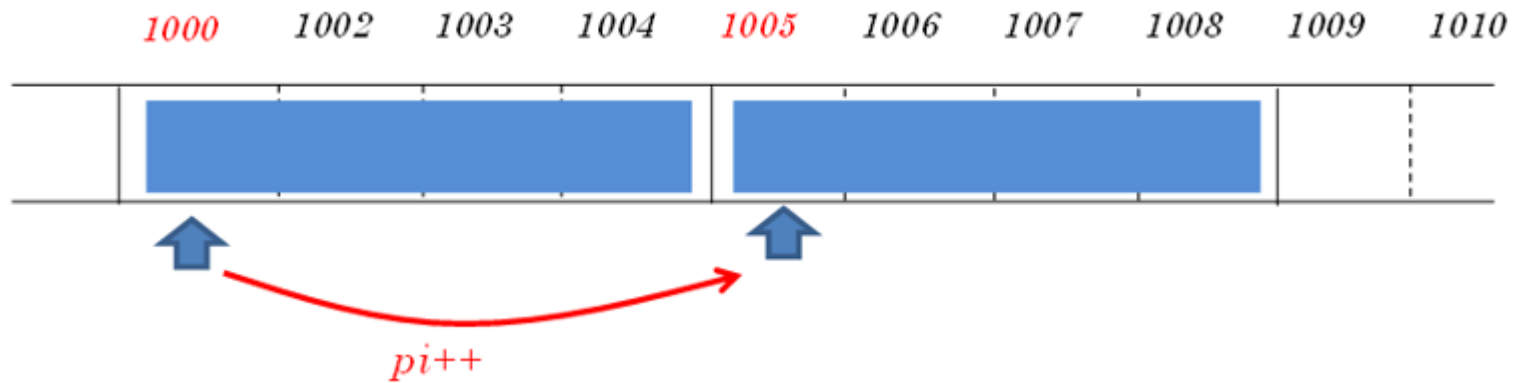
포인터의 증감 연산

```
short *ps;  
ps++;
```



포인터의 증감 연산

```
int *pi;  
pi++;
```



증가 연산

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

```
int main()
```

```
{
```

```
    int* pi;
```

```
    double* pd;
```

```
    pi = (int*) 10000;
```

```
    pd = (double*) 10000;
```

```
    cout.setf(ios::hex);
```

```
    cout << "증가전 pi: " << pi << "pd : " << pd << endl;
```

```
    pi++;
```

```
    pd++;
```

```
    cout << "증가후 pi: " << pi << "pd : " << pd << endl;
```

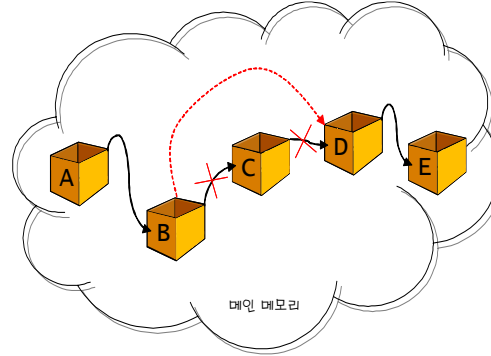
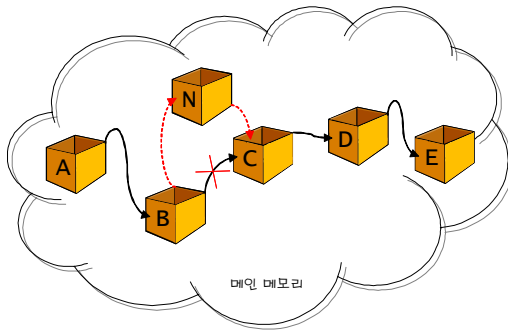
```
}
```

증가 전 pi = 10000, pd = 10000

증가 후 pi = 10004, pd = 10008

포인터 사용의 장점

- 연결 리스트나 이진 트리 등의 향상된 자료 구조를 만들 수 있다.



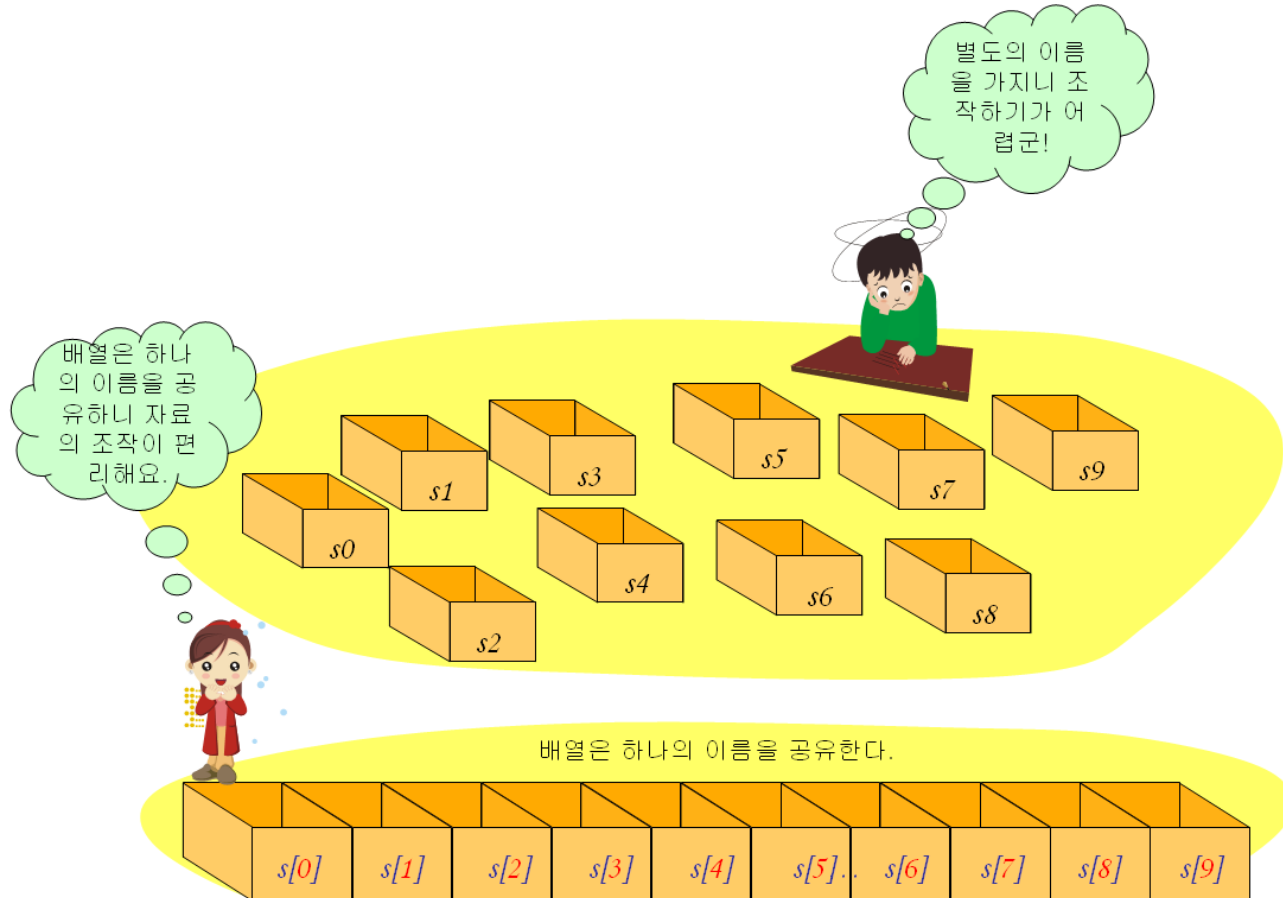
- 참조에 의한 호출
 - 포인터를 매개 변수로 이용하여 함수 외부의 변수의 값을 변경할 수 있다.
- 동적 메모리 할당

포인터와 배열

배영

배열이란?

- **배열(array)**: 동일한 타입의 데이터가 여러 개 저장되어 있는 데이터 저장 장소
- 배열 안에 들어있는 각각의 데이터들은 정수로 되어 있는 번호(첨자)에 의하여 접근
- 배열을 이용하면 여러 개의 값을 하나의 이름으로 처리할 수 있다.



배열이란?

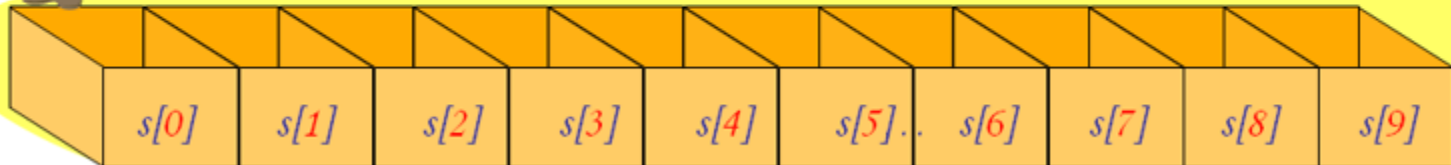


배열이란?

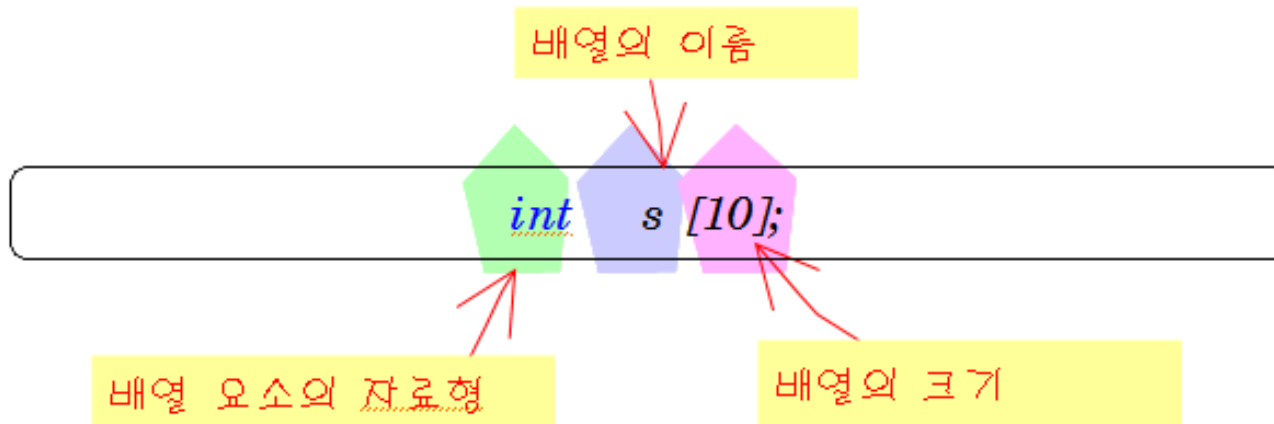
배열은 하나의 이름을 공유하니 자료의 조작이 편리해요.

```
int s[10];
```

배열은 하나의 이름을 공유한다.



배열의 선언



- 자료형: 배열 원소들이 `int`형라는 것을 의미
- 배열 이름: 배열을 사용할 때 사용하는 이름이 `s`
- 배열 크기: 배열 원소의 개수가 10개
- 인덱스(배열 번호)는 항상 0부터 시작한다.

배열 선언의 예

```
int score[60];
```

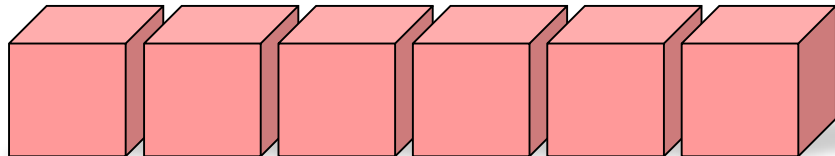
// 60개의 int형 값을 가지는 배열 s

```
float cost[12];
```

// 12개의 float형 값을 가지는 배열 cost

```
char name[50];
```

// 50개의 char형 값을 가지는 배열 name

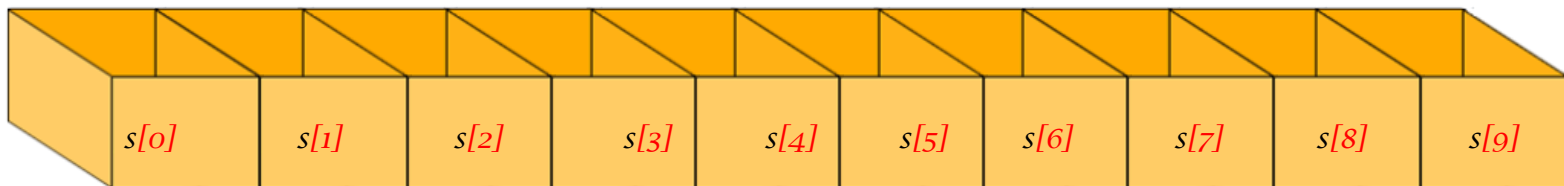


배열 원소 접근

```
int s[10];  
s[5] = 80;
```

80

인덱스

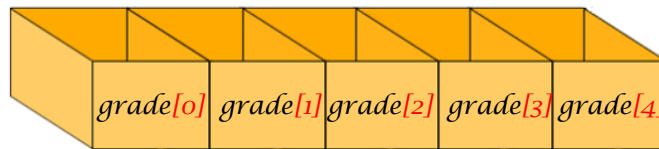


```
s[5] = 80;  
s[1] = s[0];  
s[i] = 100;      // i는 정수 변수  
s[i+2] = 100;    // 수식이 인덱스가 된다.  
s[index[3]] = 100; // index[]는 정수 배열
```

배열의 초기화

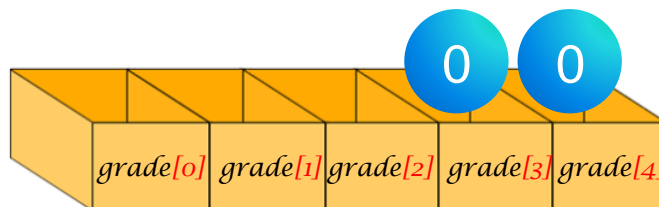
- int grade[5] = { 10,20,30,40,50 };

int grade[5] = { 10, 20, 30, 40, 50 };



- int grade[5] = { 10,20,30 };

int grade[5] = { 10, 20, 30 };



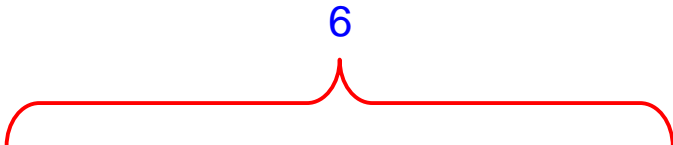
초기값을 일부만
주면 나머지
원소들은 0으로
초기화됩니다.



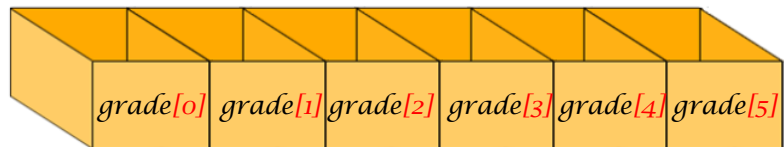
배열의 초기화

- 배열의 크기가 주어지지 않으면 자동적으로 초기값의 개수만큼이 배열의 크기로 잡힌다.

6



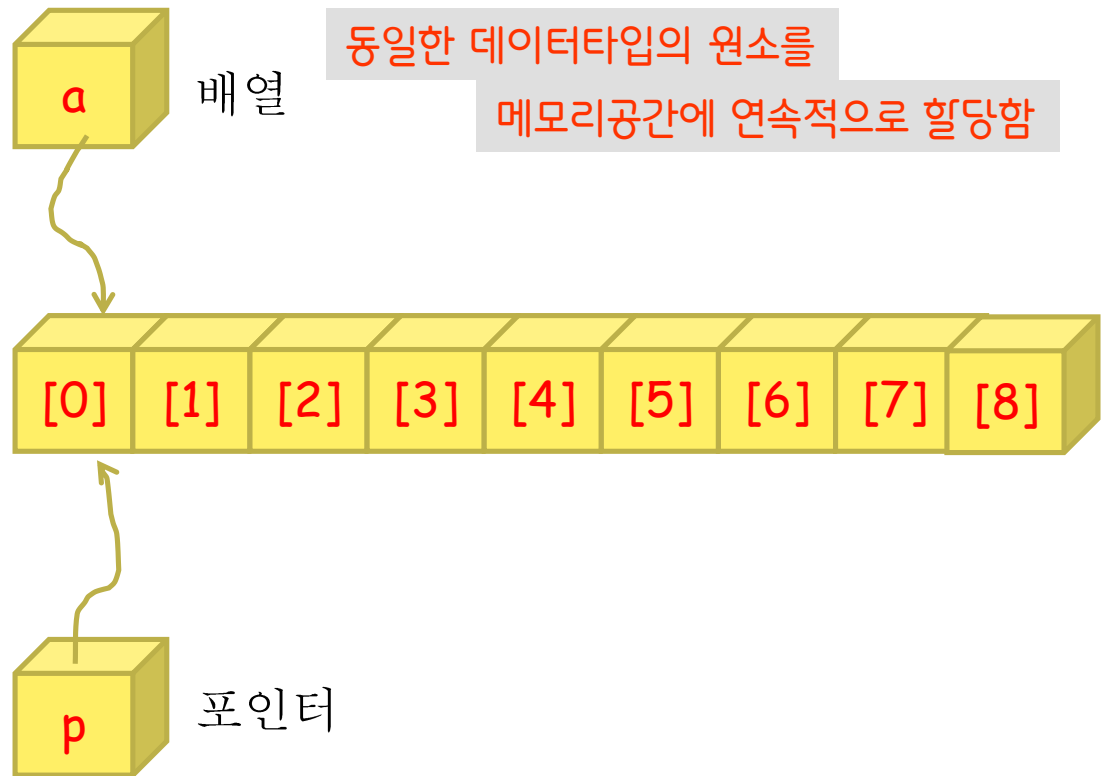
```
int grade[] = { 10, 20, 30, 40, 50, 60 };
```



포인터와 배열

포인터와 배열

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



포인터와 배열

// 포인터와 배열의 관계

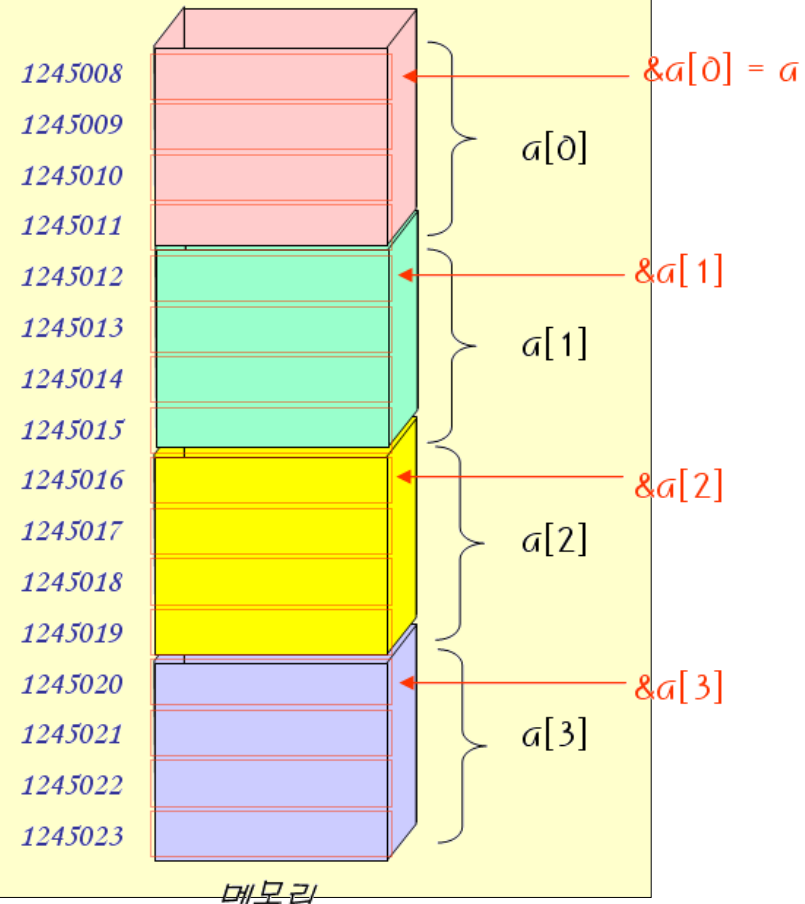
```
#include <iostream>
using namespace std;

int main(void)
{
    int a[ ] = { 10, 20, 30, 40 };

    cout << "&a[0] = " << hex << &a[0] << endl;
    cout << "&a[1] = " << hex << &a[1] << endl;
    cout << "&a[2] = " << hex << &a[2] << endl;

    cout << "배열 이름 a = " << hex << a << endl;

    return 0;
}
```



```
&a[0] = 3340364
&a[1] = 3340368
&a[2] = 3340372
a = 3340364
```

배열의 이름은 포인터?

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

int main(void)
{
    int a[4] = { 10, 20, 30, 40 };

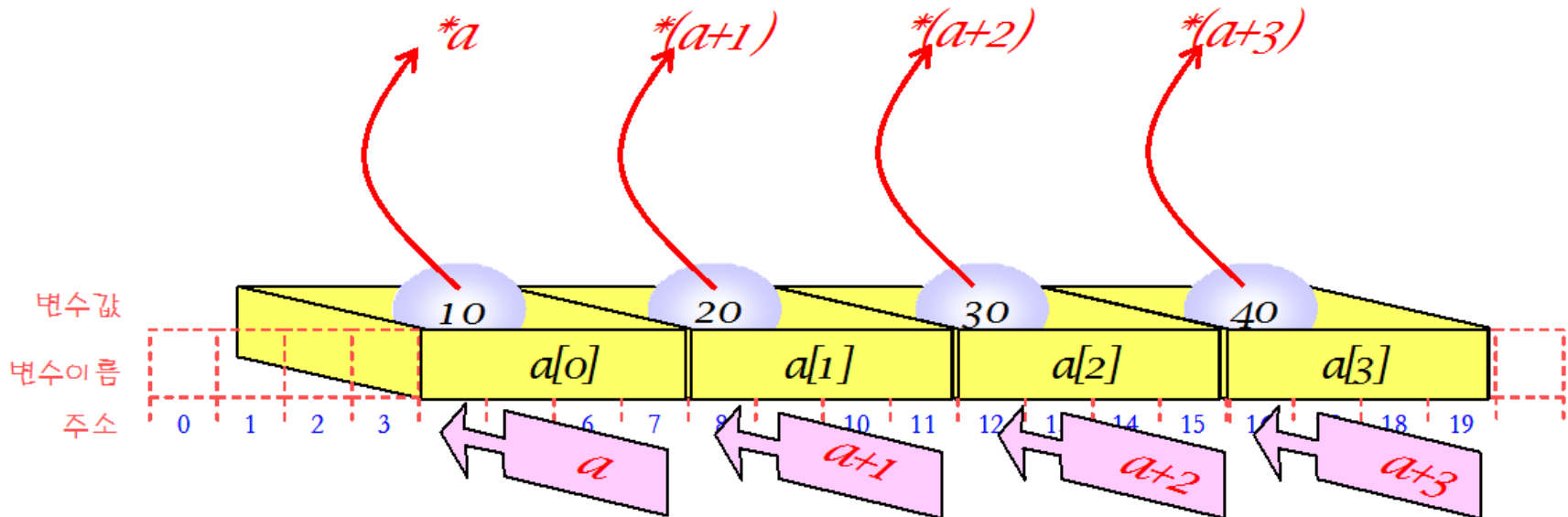
    cout << " *a = " << dec << *a << endl;
    cout << " *(a+1) = " << dec << *(a+1) << endl;
    cout << " *(a+2) = " << dec << *(a+2) << endl;

    return 0;
}
```

```
*a = 10
*(a+1) = 20
*(a+2) = 30
```

포인터와 배열

- 포인터는 배열처럼 사용할 수 있다.
- 인덱스 표기법을 포인터에 사용할 수 있다.



포인터를 배열처럼 사용

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

```
int main(void)
{
    int a[] = { 10, 20, 30, 40 };
    int* p;

    p = a;

    cout << "a[0]=" << a[0] << " a[1]=" << a[1] << " a[2]=" << a[2] << endl;
    cout << "p[0]=" << p[0] << " p[1]=" << p[1] << " p[2]=" << p[2] << endl;

    return 0;
}
```

배열은 결국 포인터로
구현된다는 것을 알 수
있다.

```
a[0]=10 a[1]=20 a[2]=30
p[0]=10 p[1]=20 p[2]=30
```

배열 원소를 포인터로 액세스

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

int main() {
    int a[3] = { 10, 20, 30 };

    for (int i = 0; i < 3; i++) cout << a[i] << endl;    //루프 ❶

    for (int i = 0; i < 3; i++) cout << *(a+i) << endl; //루프 ❷
    int* p = a;

    for (int i = 0; i < 3; i++) cout << p[i] << endl;    //루프 ❸

    for (int i = 0; i < 3; i++) cout << *(p+i) << endl; //루프 ❹

    for (int i = 0; i < 3; i++) {                          //루프 ❺
        cout << *p << endl;
        p++;
    }
}
```

모든 for 루프에서

10
20
30

출력됨

2차원 배열 이름의 포인터 타입

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

int main() {
    int a[3][2] = { 1, 2, 3, 4, 5, 6 };

    cout << "a = " << a << endl;

    cout << "a[0] = " << a[0] << endl;
    cout << "a[1] = " << a[1] << endl;
    cout << "a[2] = " << a[2] << endl;
}
```

```
a = 00F5FDE4
a[0] = 00F5FDE4
a[1] = 00F5FDEC
a[2] = 00F5FDF4
```

