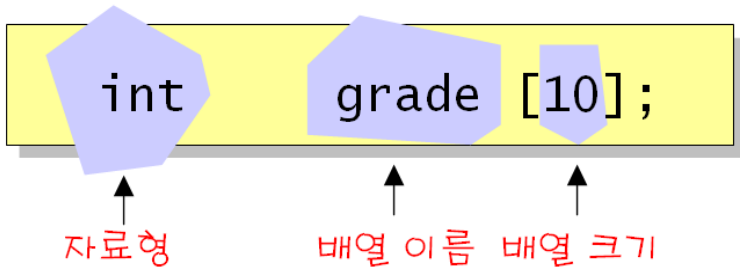


# 배열의 선언

- 배열(array): 같은 종류의 데이터들이 순차적으로 저장되어 있는 자료 구조
- 인덱스(index): 배열 원소의 번호

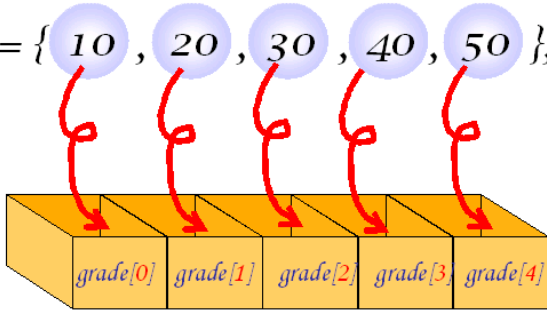


- 인덱스(배열 번호)는 항상 **?**부터 시작한다.

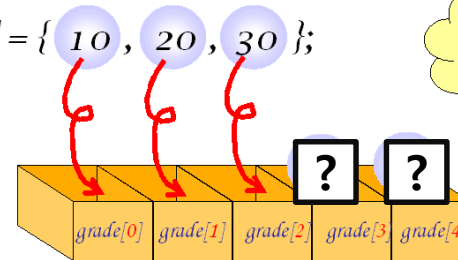
```
int score[60];           // 60개의 int형 값을 가지는 배열 grade
float cost[12];          // 12개의 float형 값을 가지는 배열 cost
char name[50];           // 50개의 char형 값을 가지는 배열 name
char src[10], dst[10];   // 2개의 문자형 배열을 동시에 선언
int index, days[7];      // 일반 변수와 배열을 동시에 선언
```

# 배열의 초기화

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



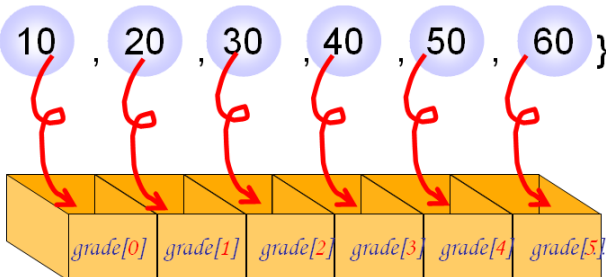
```
int grade[5] = { 10, 20, 30 };
```



초기값을 일부만 주면 나머지 원소들은 0으로 초기화됩니다.



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



배열의 크기가 주어지지 않으면  
자동적으로 초기값의 개수만큼 배열의 크기로 잡힌다.

# #실습

# 예 제

```
#include <iostream>
using namespace std;
int main(void)
{
    const int STUDENTS=5;
    int grade[STUDENTS];
    int sum = 0;
    int i, average;
    for(i = 0; i < STUDENTS; i++)
    {
        cout << "학생들의 성적을 입력하시오: ";
        cin >> grade[i];
    }
    for(i = 0; i < STUDENTS; i++)
        sum += grade[i];
    average = sum / STUDENTS;
    cout << "성적 평균= " << average << endl;
    return 0;
}
```

학생들의 성적을 입력하시오: 10  
학생들의 성적을 입력하시오: 20  
학생들의 성적을 입력하시오: 30  
학생들의 성적을 입력하시오: 40  
학생들의 성적을 입력하시오: 50  
성적 평균 = 30

# 문자열 처리 라이브러리

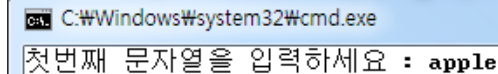
```
#include <cstring>
```

함수	설명
strlen(s)	문자열 s의 길이를 구한다.
strcpy(s1, s2)	s2를 s1에 복사한다.
strcat(s1, s2)	s2를 s1의 끝에 붙여넣는다.
strcmp(s1, s2)	s1과 s2를 비교한다.
strncpy(s1, s2, n)	s2의 최대 n개의 문자를 s1에 복사한다.
strncat(s1, s2, n)	s2의 최대 n개의 문자를 s1의 끝에 붙여넣는다.
strncmp(s1, s2, n)	최대 n개의 문자까지 s1과 s2를 비교한다.
strchr(s, c)	문자열 s안에서 문자 c를 찾는다.
strstr(s1, s2)	문자열 s1에서 문자열 s2를 찾는다.

## # 두 개의 문자열을 입력 받아, 입력 받은 내용을 출력하려고 한다.

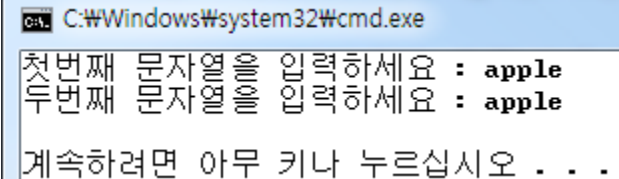
```
1  #include<iostream>
2  using namespace std;
3  int main( )
4  {
5      char str1[20];
6      char str2[20];
7      char str3[20];
8
9      cout << "첫번째 문자열을 입력하세요 : ";
10     cin >> str1;
11
12     cout << "두번째 문자열을 입력하세요 : ";
13     cin.getline(str2, 20);
14
15     cout << str1 << endl;
16     cout << str2 << endl;
17
18     return 0;
19 }
```

다음과 같은 프로그램을 실행하고 첫 번째 문자열로 apple을 입력하고 enter 를 입력했다.



C:\Windows\system32\cmd.exe  
첫번째 문자열을 입력하세요 : apple

Q. Enter를 입력한 후 발생하는 일은?



C:\Windows\system32\cmd.exe  
첫번째 문자열을 입력하세요 : apple  
두번째 문자열을 입력하세요 : apple  
계속하려면 아무 키나 누르십시오 . . .

**WHY?**

**cin.getline() :**

공백, 개행도 입력으로 받음

## 문자열 처리 라이브러리 연습해보기 – strlen(s)

- 문자열 s의 길이를 구한다
- Null문자를 제외한 길이를 계산

**실습 : 문자열을 입력받아 str1에 저장하고, 그 길이를 출력하라**

```
1 #include<iostream>
2 #include<string>
3 using namespace std;
4 int main()
5 {
6     char str1[20];
7
8     cout << "str1 문자열을 입력하세요 : ";
9     cin >> str1;
10
11     cout << "str1의 길이: " <<          << endl;
12
13     return 0;
14 }
```

C:\Windows\system32\cmd.exe

```
str1 문자열을 입력하세요 : appleapple
str1의 길이: 10
계속하려면 아무 키나 누르십시오 . . .
```

## 문자열 처리 라이브러리 연습해보기 – strcpy(s1, s2)

- 문자열 s2를 s1에 복사한다.
- 복사전 메모리 공간을 확인하지 않아 오버플로우 발생가능

**실습 : 문자열을 입력받아 str1에 저장하고,**


- str2에 str1을 복사한다.
- Str3에 문자열 "복사할거얌"을 복사한다.

```
1  #include<iostream>
2  #include<string>
3  using namespace std;
4  int main()
5  {
6      char str1[20];
7      char str2[20];
8      char str3[20];
9
10     cout << "str1 문자열을 입력하세요 : ";
11     cin >> str1;
12
13     
14
15
16     cout << "str2: " << str2 << endl;
17     cout << "str3: " << str3 << endl;
18
19     return 0;
20 }
```

## 문자열 처리 라이브러리 연습해보기 – strcat(s1, s2)

- 문자열 s2를 s1의 끝에 붙여넣는다.
- S2의 시작 문자가 s1의 마지막에 있는 Null문자를 덮어쓰며 두 개의 문자열을 연결
- 복사전 메모리 공간을 확인하지 않아 오버플로우 발생가능

```
C:\Windows\system32\cmd.exe
str1 문자열을 입력하세요 : pine
str2 문자열을 입력하세요 : apple
str1: pineapple
계속하려면 아무 키나 누르십시오 . . .
```

```
1 #include<iostream>
2 #include<string>
3 using namespace std;
4 int main()
5 {
6     char str1[20];
7     char str2[20];
8
9     cout << "str1 문자열을 입력하세요 : ";
10    cin >> str1;
11    cout << "str2 문자열을 입력하세요 : ";
12    cin >> str2;
13
14    
15
16    cout << "str1: " << str1 << endl;
17
18    return 0;
19 }
```



## 문자열 처리 라이브러리 연습해보기 – strcmp(s1, s2)

- 문자열 s1과 s2를 비교한다.

```
1 #include<iostream>
2 #include<cstring>
3 using namespace std;
4 int main()
5 {
6     char str1[20];
7     char str2[20];
8
9     cout << "str1 문자열을 입력하세요 : ";
10    cin >> str1;
11    cout << "str2 문자열을 입력하세요 : ";
12    cin >> str2;
13
14    int result = strcmp(str1, str2);
15
16    if ( )
17        cout << "같은 문자열" << endl;
18    else if ( )
19        cout << "str1이 str2보다 사전적으로 앞에 있음" << endl;
20    else
21        cout << "str1이 str2보다 사전적으로 뒤에 있음" << endl;
22
23    return 0;
24 }
```

반환 값	string1과 string2의 관계
음수 (<0)	문자열 string1이 string2보다 사전적으로 앞에 있습니다.
0	문자열 string1과 string2는 일치합니다.
양수 (>0)	문자열 string1이 string2보다 사전적으로 뒤에 있습니다.

```
C:\Windows\system32\cmd.exe
str1 문자열을 입력하세요 : zoo
str2 문자열을 입력하세요 : apple
str1이 str2보다 사전적으로 뒤에 있음
계속하려면 아무 키나 누르십시오 . . .
```

# #실습

## 예제

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

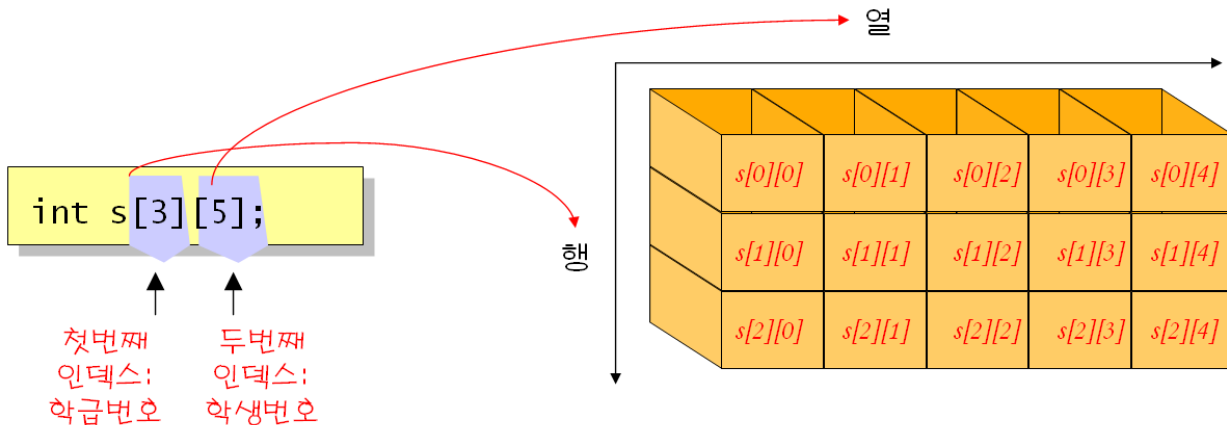
Hello World from strcpy() and strcat()!

```
int main()
{
    char string[80];

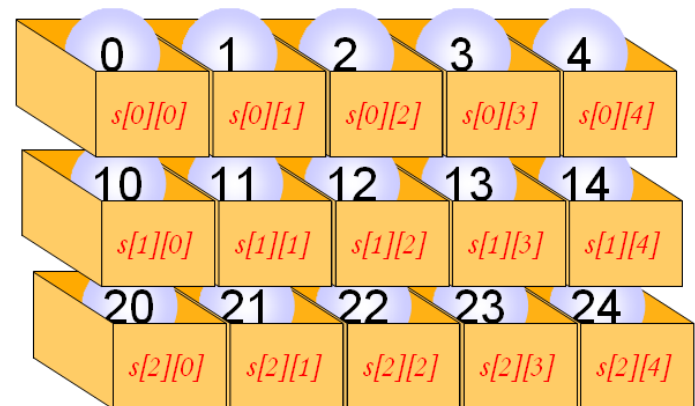
    strcpy( string, "Hello World from " );
    strcat( string, "strcpy() " );
    strcat( string, "and " );
    strcat( string, "strcat()!" );
    cout << string << endl;
    return 0;
}
```

# 2차원 배열

```
int s[10];    // 1차원 배열  
int s[3][10]; // 2차원 배열  
int s[5][3][10]; // 3차원 배열
```

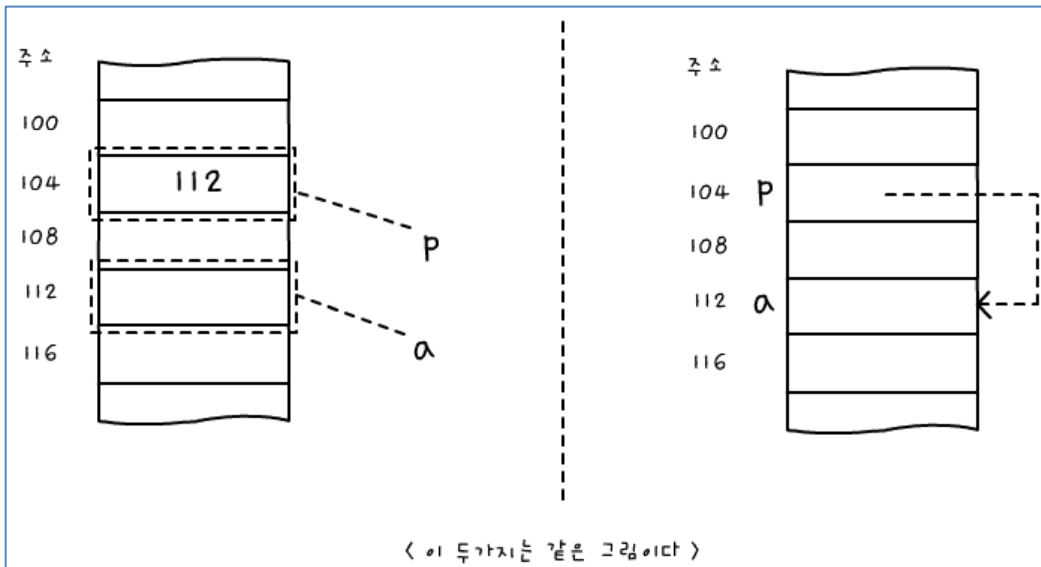


```
int s[ ][5] = {  
    { 0, 1, 2, 3, 4}, // 첫 번째 행의 원소들의 초기값  
    { 10, 11, 12, 13, 14}, // 두 번째 행의 원소들의 초기값  
    { 20, 21, 22, 23, 24}, // 세 번째 행의 원소들의 초기값  
};
```



# 포인터란?

- 포인터 변수는 다른 변수를 가리키는 변수다.
  - ▣ 포인터 변수에는 다른 변수의 주소 값을 저장할 수 있다.
  - ▣ 예를 들어서 포인터 변수 p가 다른 변수 a의 주소 값을 보관하고 있다면, 포인터 변수 p가 변수 a를 가리키고 있다고 말한다.



포인터를 이용하여 메모리의 내용에 직접 접근할 수 있다!



# 간접 참조 연산자

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

```
int i = 10;
int *p = &i;

cout << *p; // 10이 출력된다.

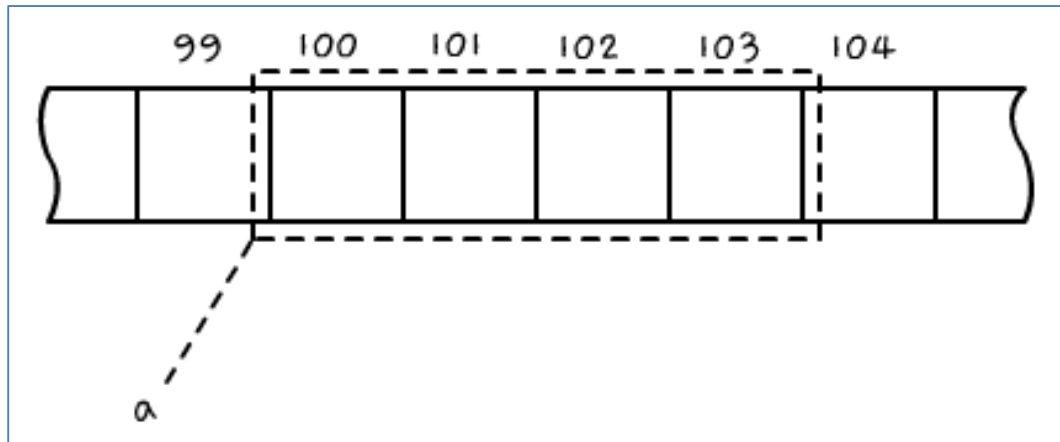
*p = 20;
cout << *p; // 20이 출력된다.
```

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

포인터타입	++ 후 값
char	1
short	2
int	4
float	4
double	8

# 변수의 주소

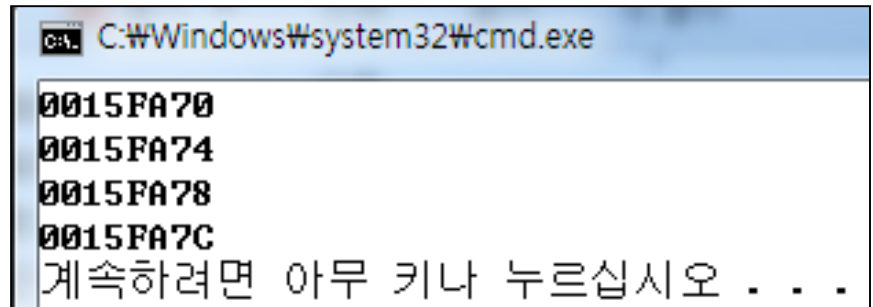
- 변수의 주소란 변수가 포함하고 있는 **제일 첫 번째 바이트의 주소**를 말한다.
  - ▣ 예를 들어서 int 타입의 변수 a가 있다고 하면, 변수 a는 4바이트의 메모리 공간을 소유하게 된다.
  - ▣ 메모리의 각 바이트마다 주소가 있다.
  - ▣ 그 중에서 첫번째 바이트의 주소를 변수 a의 주소라고 말한다.



## #실습

## 증가 연산 예제

```
1  #include<iostream>
2  using namespace std;
3  int main( )
4  {
5      int a = 100;
6      int *p = &a;
7
8      cout << p << endl;
9      cout << ++p << endl;
10     cout << ++p << endl;
11     cout << ++p << endl;
12
13     return 0;
14 }
```



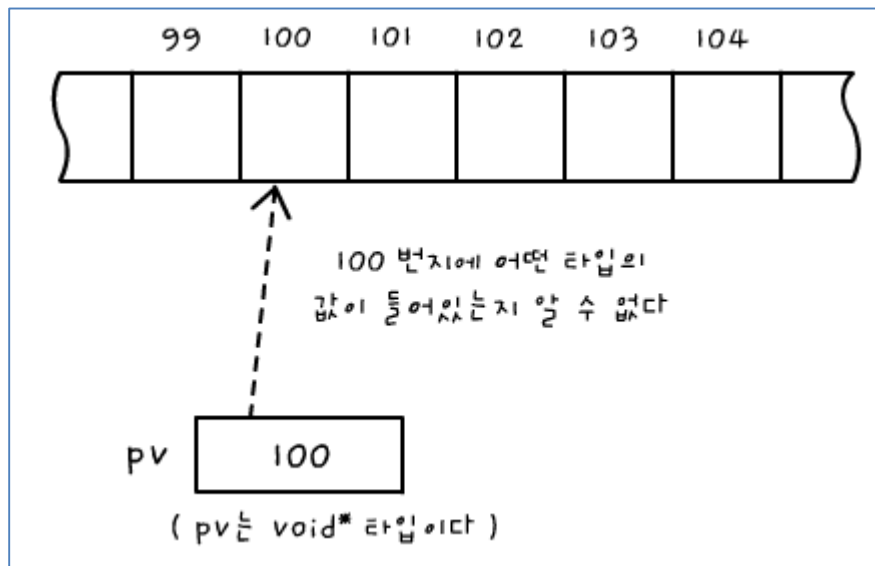
C:\Windows\system32\cmd.exe

0015FA70  
0015FA74  
0015FA78  
0015FA7C  
계속하려면 아무 키나 누르십시오 . . .

# void 포인터

- void\* 타입의 포인터 변수는 어떤 타입의 변수라도 가리킬 수 있다.
- 그러므로, void 포인터가 가리키는 데이터의 크기나 종류를 알아낼 수 없다. ( 개발자 스스로 기억하고 있어야 한다. )

```
void* p;
```





# void 포인터

- 17 • C++에서는 묵시적인 포인터 변환은 허용되지 않는다.

```
int *pi;  
double *pd;  
void* vp;
```

```
vp=pi;  
pd=vp;
```

결과적으로 int 포인터가  
double 포인터에 대입됨

1>c:\Users\Westher\documents\visual studio 2008\projects\test\test\vp.cpp(9) : error C2440: '=' : 'void \*'에서'double \*'(으)로 변환 할 수 없습니다.

1>'void\*'에서 'void'가 아닌 포인터로 변환하려면 명시적 캐스트가 필요합니다.

1>빌드 로그가 "file://c:\Users\ESTHER\Documents\Visual Studio 2008\Projects\test\test\Debug\BuildLog.htm"에 저장되었습니다.

1>test- 오류: 1개, 경고: 0개

===== 빌드: 성공0, 실패1, 최신0, 생략0 =====

# 주소를 사용해서 정보에 접근하기

- 포인터 변수가 가리키는 변수에 접근하는 예

```
// p가 a를 가리키도록 만든다.
```


```
int a = 123;
```

```
int* p = &a;
```

```
// p가 가리키는 변수의 값을 얻는다.
```

```
cout << "p = " << *p << "\n";
```

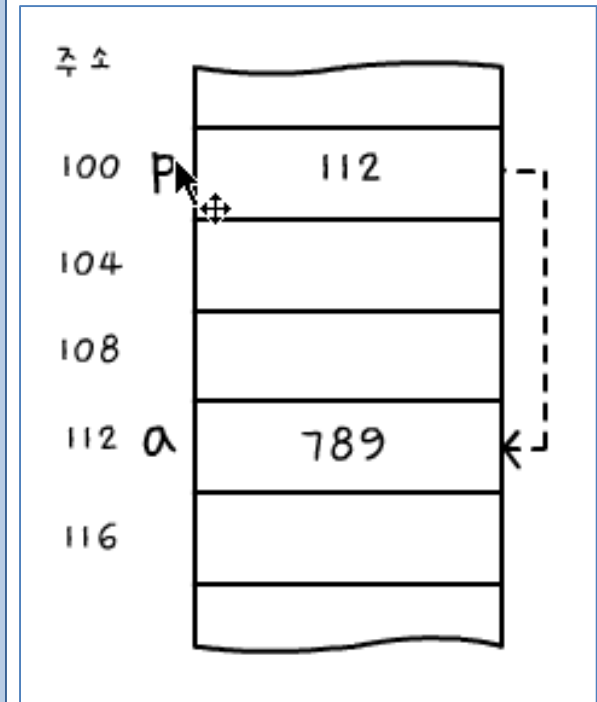
```
// p가 가리키는 변수의 값을 변경한다.
```

```
 = 789;
```

```
// 관련 정보를 출력한다.
```

```
cout << "a = " << a << "\n";
```

```
cout << "p = " << *p << "\n";
```



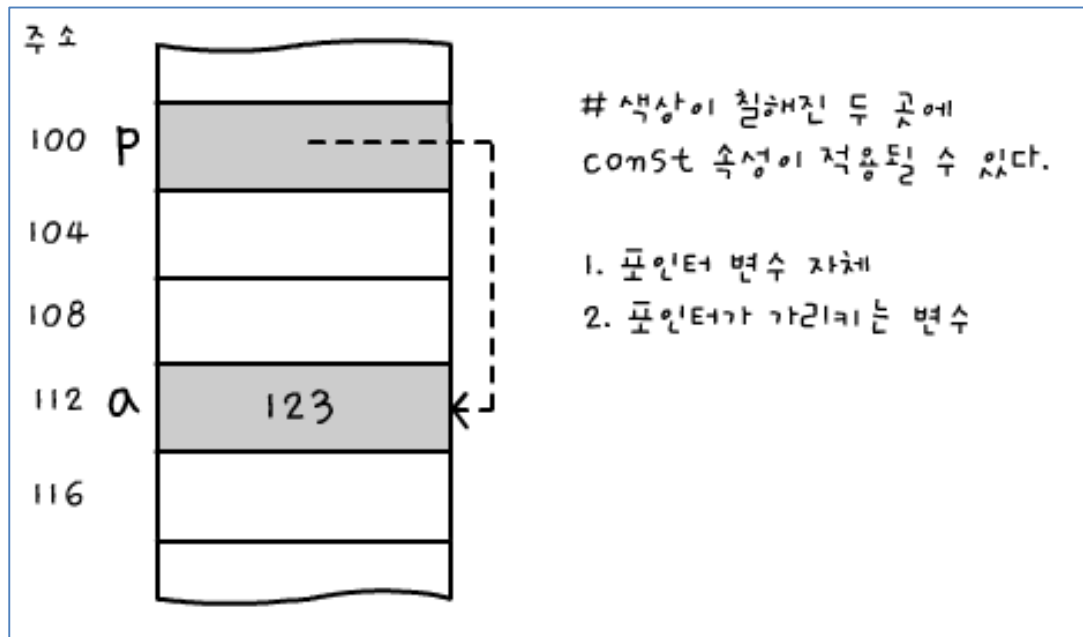
# 포인터 안전하게 사용하기

- 잘못된 주소를 가진 포인터를 사용하는 것은 매우 위험하기 때문에, 포인터를 사용할 때는 다음의 가이드 라인을 따른다.
  - 포인터 변수는 항상 0 혹은 NULL 값으로 초기화 한다.
  - 포인터 변수를 사용하기 전에는 0 혹은 NULL 값을 가지고 있는지 확인 한다.

```
// 포인터 변수를 정의하고 초기화한다.  
int* p = NULL;  
  
// 이 상태에서 포인터를 사용해보자.  
if (NULL != p)  
    *p = 30;  
  
// p가 변수를 가리키게 만들자  
int a = 100;  
p = &a;  
  
// 이 상태에서 포인터를 사용해보자.  
if (!p)  
    *p = 30;
```

# Const 와 포인터 (1)

- 포인터 변수에 Const 속성을 부여할 때는 다음의 두 변수를 고려할 수 있다.
  - 포인터 변수 자체
  - 포인터 변수가 가리키는 변수



## Const 와 포인터 (2)

- 포인터 변수에 Const 속성을 적용한 3가지 경우를 비교해보자.
  - 포인터 변수가 가리키는 변수가 const인 경우

```
int i1 = 10;  
int i2 = 20;  
const int* p = &i1;  
  
p = &i2;    // OK  
*p = 30;    // FAIL
```

- 포인터 변수 자신이 const인 경우

```
int i1 = 10;  
int i2 = 20;  
int* const p = &i1;  
  
p = &i2;    // FAIL  
*p = 30;    // OK
```

- 두 변수 모두 const인 경우

```
int i1 = 10;  
int i2 = 20;  
const int* const p = &i1;  
  
p = &i2; // FAIL  
*p = 30; // FAIL
```

# #실습

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

void display(const int *xpos, const int *ypos);
void move(int *xpos, int *ypos);
```

```
int main(void)
{
    int x = 10;
    int y = 20;

    display(&x, &y);
    move(&x, &y);
    display(&x, &y);

    return 0;
}
```

```
void display(const int *xpos, const int *ypos)
{
    cout<<"현재의 위치"<< *xpos<< *ypos<<endl;
}
```

```
void move(int *xpos, int *ypos)
{
    *xpos = *xpos + 1;
    *ypos = *ypos + 1;
    • }
```

매개변수 앞에 const를 붙이면 매개변수를 통하여 주소가 가리키는 메모리값을 변경할 수 없다. 참조만 할 수 있다.

# #실습

## 포인터와 배열

- 배열 이름은 첫 번째 배열 원소의 주소
- 포인터를 사용하여서 배열 원소를 처리 가능

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

10 20 30 40 50

```
int main(void)
```

```
{
```

```
    const int STUDENTS = 5;
```

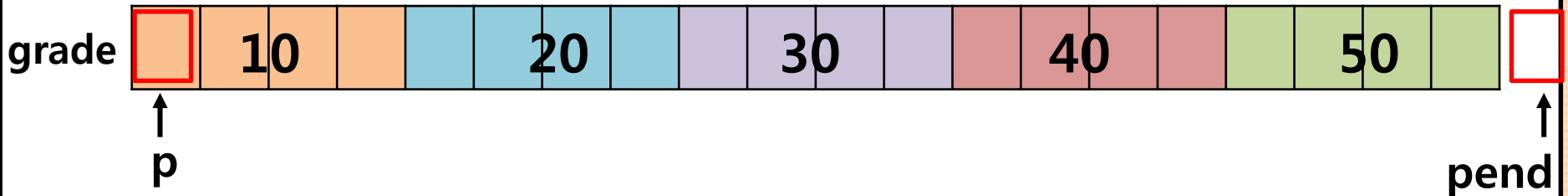
```
    int grade[STUDENTS] = { 10, 20, 30, 40, 50 };
```

```
    for ( int i = 0; i < STUDENTS; i++ )
```

```
        cout << grade[i] << " ";
```

```
    return 0;
```

```
}
```



# #실습

## 배열과 함수

```
#include <iostream>
```

```
int get_average(const int score[], int n);  
void increment(int score[], int n);
```

```
int main(void)  
{
```

```
    const int STUDENTS=5;  
    int grade[STUDENTS] = { 1, 2, 3, 4, 5 };  
    int avg;
```

```
    increment(grade, STUDENTS);  
    avg = get_average(grade, STUDENTS);  
    cout << "평균은 " << avg << "입니다." << endl;
```

```
    return 0;
```

```
}  
void increment(int score[], int n)  
{  
    int i;  
  
    for(i = 0; i < n; i++)  
        ++score[i];  
}
```

- 배열을 함수의 인수로 전달하는 경우에는 배열 이름이 포인터이므로 **포인터가 전달된다.**

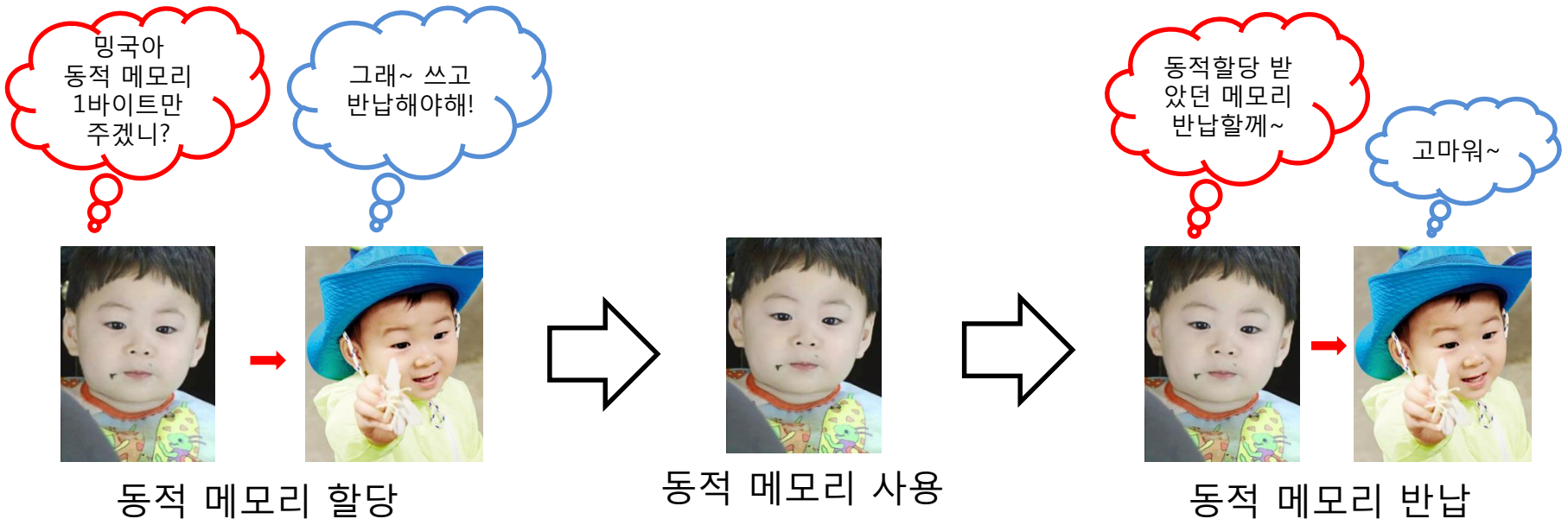
```
int get_average(const int score[], int n)    // ②  
{  
    int i;  
    int sum = 0;  
  
    for(i = 0; i < n; i++)  
        sum += score[i];  
  
    return sum / n;  
}
```



# 동적 메모리

## • 동적 메모리

- 실행 도중에 동적으로 메모리를 할당받는 것
- 사용이 끝나면 시스템에 메모리를 반납
- 필요한 만큼만 할당을 받고 메모리를 매우 효율적으로 사용
- **new**와 **delete** 키워드 사용



# 동적 메모리 할당의 과정1

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

int main()
{
    int *pi;           // 동적 메모리를 가리키는 포인터

    pi = new int[100]; // ① 동적 메모리 할당

    for(int i=0; i < 100; i++)
        *(pi+i) = 0;   // ② 동적 메모리 사용

    delete[] pi;       // ③ 동적 메모리 반납

    return 0;
}
```

## #실습

# 동적 메모리 할당의 과정2

```
1  #include<iostream>
2  #include<string.h>
3  #include<stdlib.h>
4  using namespace std;
5  char* MakeStrAdr(int len)
6  {
7      //char* str = (char*)malloc(sizeof(char)*len);
8      char* str = new char[len];
9      return str;
10 }
11
12 int main(void)
13 {
14     char* str = MakeStrAdr(20);
15     strcpy(str, "I am so happy!!");
16     cout << str << endl;
17     //free(str);
18     delete[]str;
19
20     return 0;
21 }
```

**I am so happy!!**

계속하려면 아무 키나 누르십시오 . . .

# 참조자

- 참조자(reference): 변수에 별명을 붙이는 것  
**int &ref = var;** : "참조자 ref은 변수 var의 별명(alias)이다"
- 객체의 크기가 큰 경우, 복사는 시간이 많이 걸린다. 이때는 참조자로 처리하는 것이 유리

# #실습

# 예제

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

```
int main()
{
```

```
    int var;
```

```
    int &ref = var;           // 참조자선언
```

```
    var = 10;
```

```
    cout << "var의 값: " << var << endl;
```

```
    cout << "ref의 값: " << ref << endl;
```

```
    ref = 20;                // ref의 값을 변경하면 var의 값도 변경된다.
```

```
    cout << "var의 값: " << var << endl;
```

```
    cout << "ref의 값: " << ref << endl;
```

```
    return 0;
```

```
}
```

var의 값: 10  
ref의 값: 10  
var의 값: 20  
ref의 값: 20

# 포인터 vs 참조자

- 일반적으로 참조자를 사용하는 편이 쉽다.
- 만약 참조하는 대상이 **수시로 변경**되는 경우에는 포인터를 사용
- **NULL이 될 가능성**이 있는 경우에도 포인터를 사용

## # 참조자와 포인터

- 참조자는 반드시 선언과 동시에 초기화  
`int &ref;` // 오류!
- 포인터는 변경될 수 있지만 참조자는 변경이 불가능하다.  
`int &ref = var1;`  
`ref = var2;` // 오류!
- 참조자를 상수로 초기화하면  
`int &ref = 10;` // 오류!

# 포인터와 함수

- C++에서의 인수 전달 방법
  - 값에 의한 호출 (call-by-value)
    - 함수로 복사본이 전달된다.
  - 참조에 의한 호출 (call-by-reference)
    - 함수로 원본이 전달된다.
    - 포인터를 이용한 호출 & 참조자를 이용한 호출

# #실습

## 참조에 의한 호출(참조자)

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

```
void swap(int &rx, int &ry);
int main()
```

```
{
    int a = 100, b = 200;
    cout << "swap() 호출전: a = " << a << ", b = " << b << endl;
    swap(a, b);
    cout << "swap() 호출후: a = " << a << ", b = " << b << endl;
    return 0;
}
```

```
void swap(int &rx, int &ry)
```

```
{
    int tmp;

    tmp = rx;
    rx = ry;
    ry = tmp;
}
```

swap() 호출전: a = 100, b = 200  
swap() 호출후: a = 200, b = 100