

## 9장. 포인터

숭실대학교  
전자정보공학부 IT융합전공  
담당교수: 권민혜  
minhae@ssu.ac.kr

# 이번 장에서 학습할 내용

- 포인터의 개념을 이해한다.
- 포인터 선언 및 초기화 과정을 이해한다.
- 포인터의 연산의 특수성을 이해한다.
- 포인터와 배열의 관계를 이해한다.
- 포인터를 이용한 참조에 의한 호출을 이해한다

# 이번 장에서 만들 프로그램

- 변수의 주소를 계산하는 프로그램을 작성해보자.

```
Microsoft Visual Studio 디버그 콘솔
i의 주소: 012FFA0C
c의 주소: 012FFA03
f의 주소: 012FF9F0
```

- 변수 a와 b의 내용을 서로 바꾸는 함수 swap()을 작성해보자.

```
Microsoft Visual Studio 디버그 콘솔
swap() 호출전 a=100 b=200
swap() 호출후 a=200 b=100
```

- 배열을 처리하는 함수들을 정의해서 사용해보자.

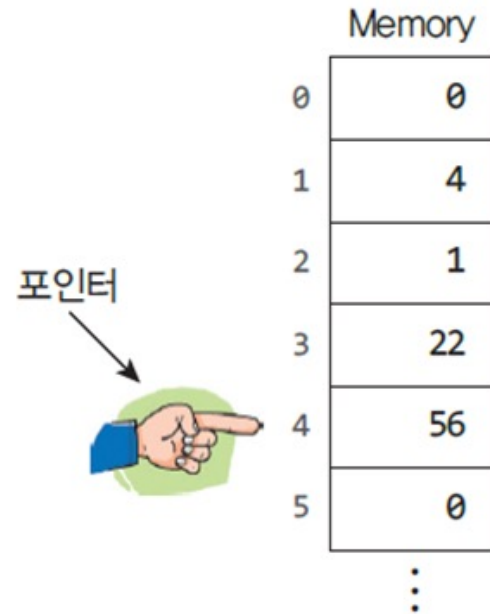
```
Microsoft Visual Studio 디버그 콘솔
[ 10 20 30 40 50 ]
get_array_avg() 호출
배열 원소들의 평균 = 30.000000
```

# 포인터란?

- *포인터(pointer)*: 주소를 가지고 있는 변수

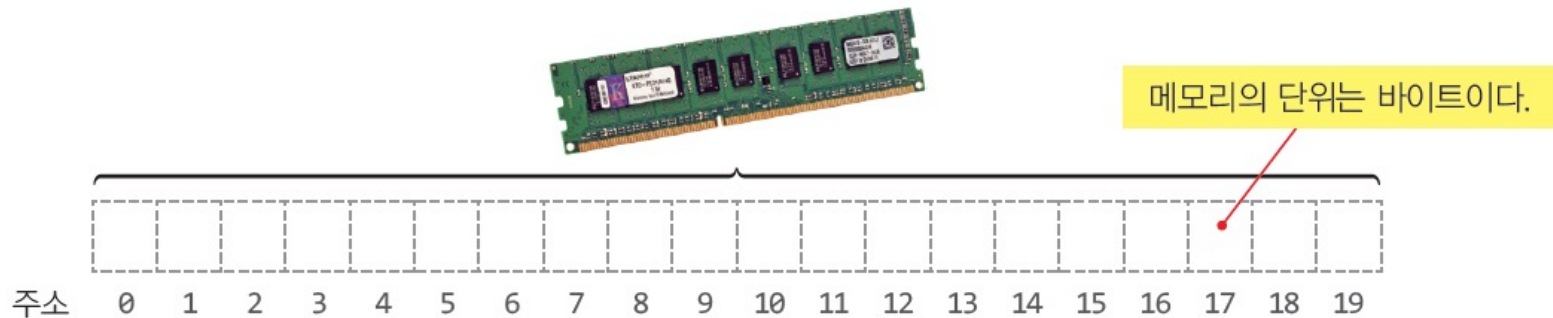


집



# 메모리의 구조

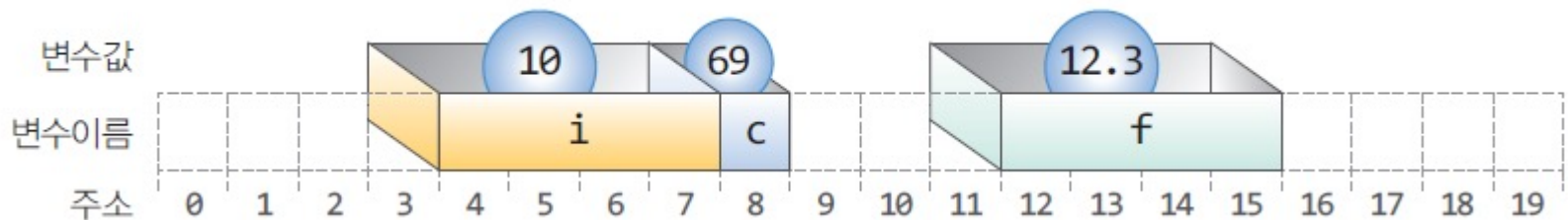
- 변수는 메모리에 저장된다.
- 메모리는 바이트 단위로 액세스된다.
- 첫번째 바이트의 주소는 0, 두번째 바이트는 1,...



# 변수와 메모리

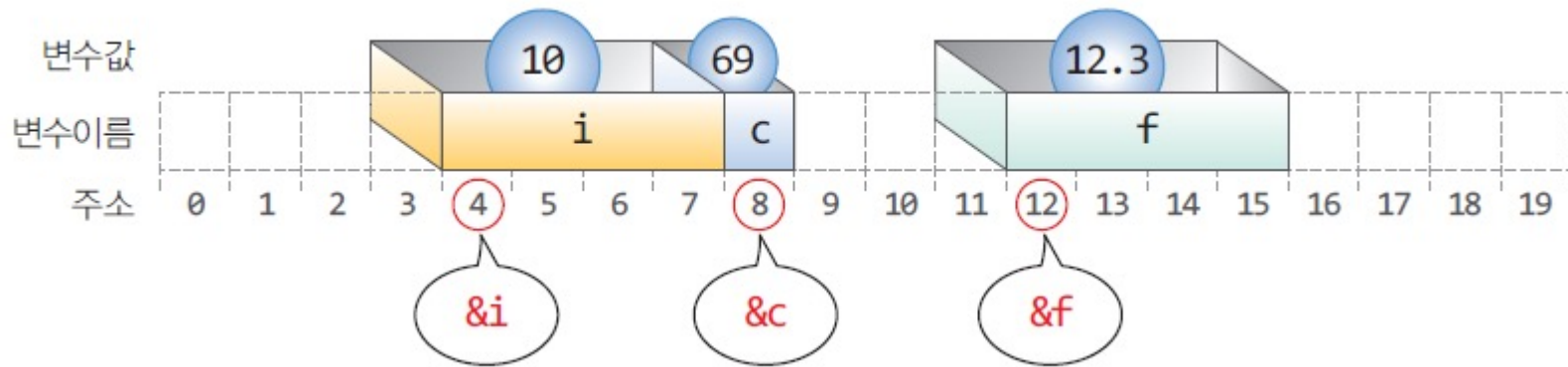
- 변수의 크기에 따라서 차지하는 메모리 공간이 달라진다.
- char형 변수: 1바이트, int형 변수: 4바이트,...

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



# 변수의 주소

- 변수의 주소를 계산하는 연산자: &
- 변수 i의 주소: &i



# 변수의 주소

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

    printf("i의 주소: %u\n", &i);           // 변수 i의 주소 출력
    printf("c의 주소: %u\n", &c);           // 변수 c의 주소 출력
    printf("f의 주소: %u\n", &f);           // 변수 f의 주소 출력
    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

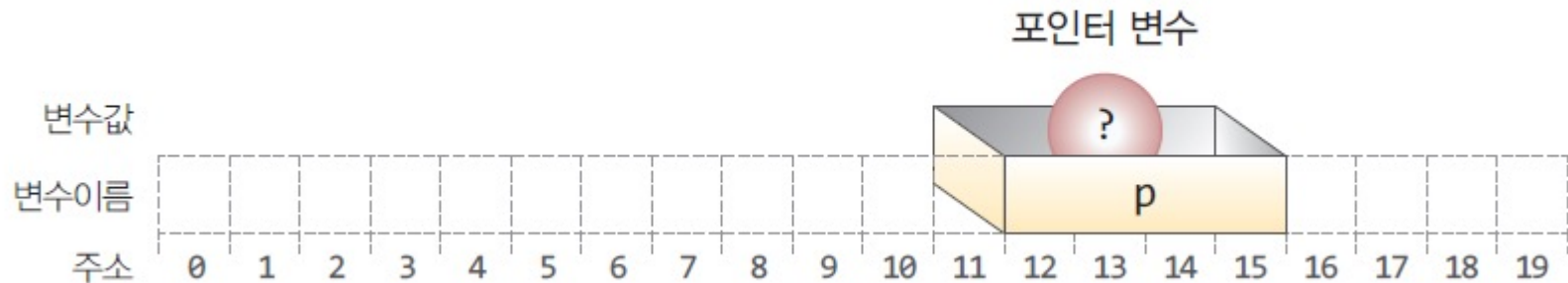
```
i의 주소: 012FFA0C
c의 주소: 012FFA03
f의 주소: 012FF9F0
```



# 포인터의 선언

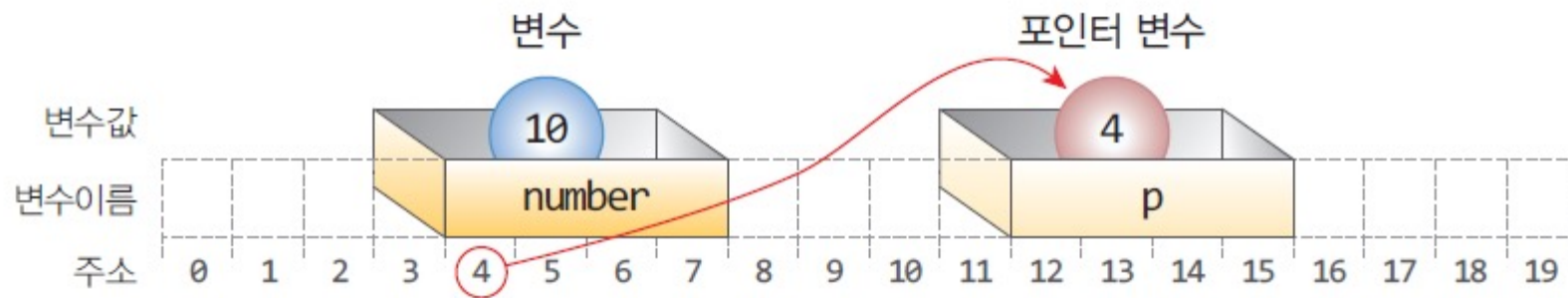
- 포인터: 변수의 주소를 가지고 있는 변수

```
int *p
```



# 포인터와 변수의 연결

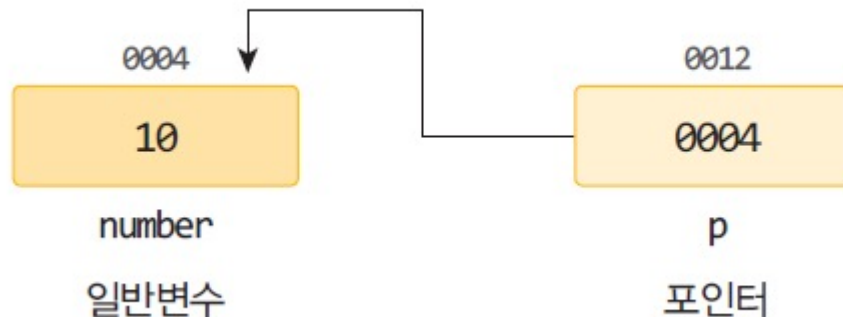
```
int    number = 10;      // 정수형 변수 number 선언
int    *p;               // 포인터 변수 p 선언
p = &number;             // 변수 number의 주소가 포인터 p로 대입
```



# 포인터와 변수

- 포인터 `p`가 변수 `number`를 가리킨다.

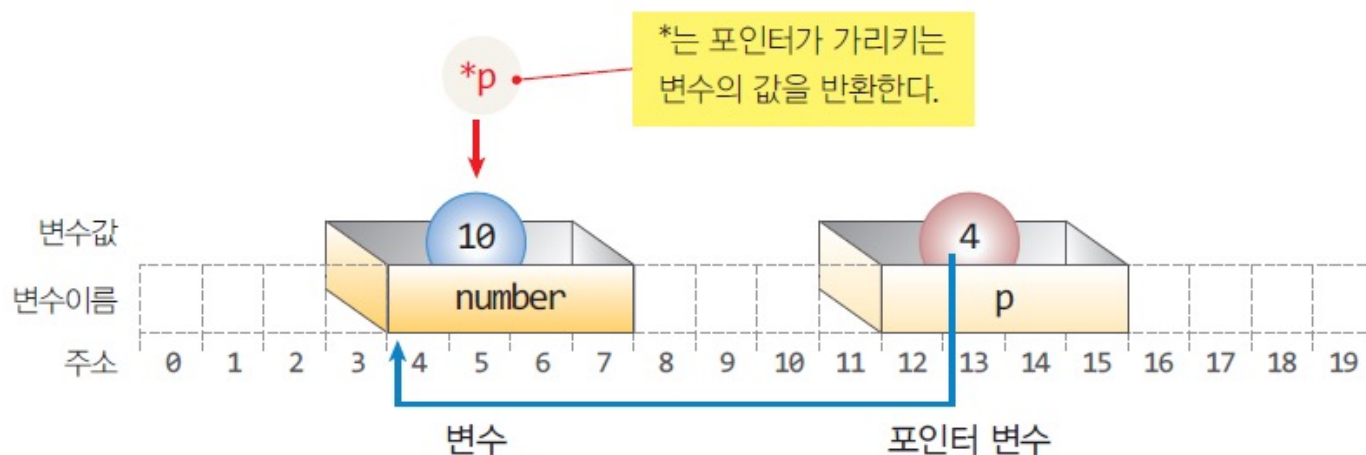
```
int    number = 10;    // 정수형 변수 number 선언
int    *p;             // 포인터 변수 p 선언
p = &number;           // 변수 number의 주소가 포인터 p로 대입
```



## 간접 참조 연산자

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

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



# 포인터 연산자

- 포인터에 관련된 연산자는 다음과 같은 2가지이다.



# 예제 #1

%p: Hexadecimal value로 display해줌  
(memory 주소 프린트시 자주 사용)

```
#include <stdio.h>

int main(void)
{
    int number = 10;
    int* p;

    p = &number;

    printf("변수 number의 주소 = %p\n", &number);
    printf("포인터의 값 = %p\n", p);
    printf("변수 number의 값 = %d\n", number);
    printf("포인터가 가리키는 값 = %d\n", *p);

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

```
변수 number의 주소 = 006FFB90
포인터의 값 = 006FFB90
변수 number의 값 = 10
포인터가 가리키는 값 = 10
```

## 예제 #2

```
#include <stdio.h>

int main(void)
{
    int number = 10;
    int* p;

    p = &number;
    printf("변수 number의 값 = %d\n", number);

    *p = 20;
    printf("변수 number의 값 = %d\n", number);

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

변수 number의 값 = 10  
변수 number의 값 = 20

# 중간 점검



## 중간점검

1. 메모리는 어떤 단위를 기준으로 주소가 매겨지는가?
2. 포인터도 변수인가?
3. 변수의 주소를 추출하는데 사용되는 연산자는 무엇인가?
4. 변수  $x$ 의 주소를 추출하여 변수  $p$ 에 대입하는 문장을 쓰시오.
5. 정수형 포인터  $p$ 가 가리키는 위치에 25를 저장하는 문장을 쓰시오.





# 포인터 연산

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

$++p;$

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

포인터의 증가는  
일반 변수와는 약간  
다릅니다. 가리키는 객체의  
크기만큼 증가합니다.



# 증가 연산 예제

```
#include <stdio.h>
int main(void)
{
    char *pc;
    int *pi;
    double *pd;
    // 절대주소 대입(좋지않은 습관, 예제용으로만 보기)
    pc = (char *)10000;
    pi = (int *)10000;
    pd = (double *)10000;
    printf("증가 전 pc = %d, pi = %d, pd = %d\n", pc, pi, pd);

    pc++;
    pi++;
    pd++;
    printf("증가 후 pc = %d, pi = %d, pd = %d\n", pc, pi, pd);

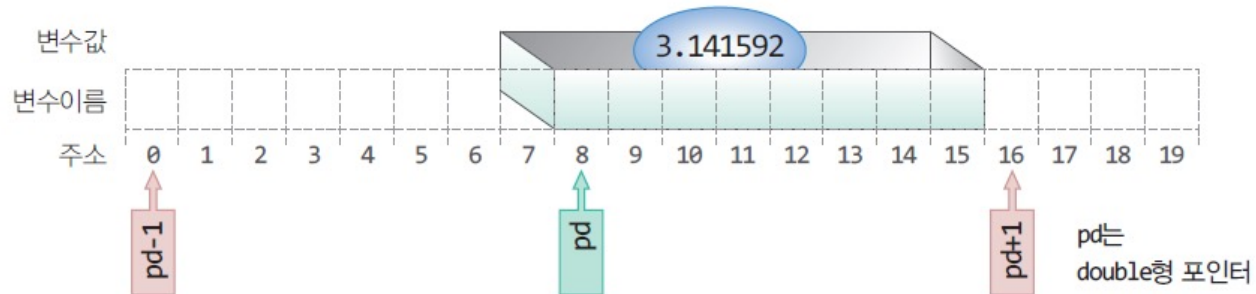
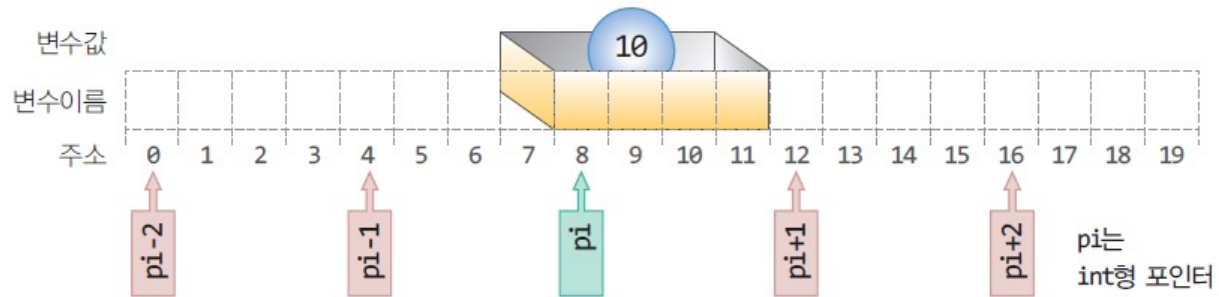
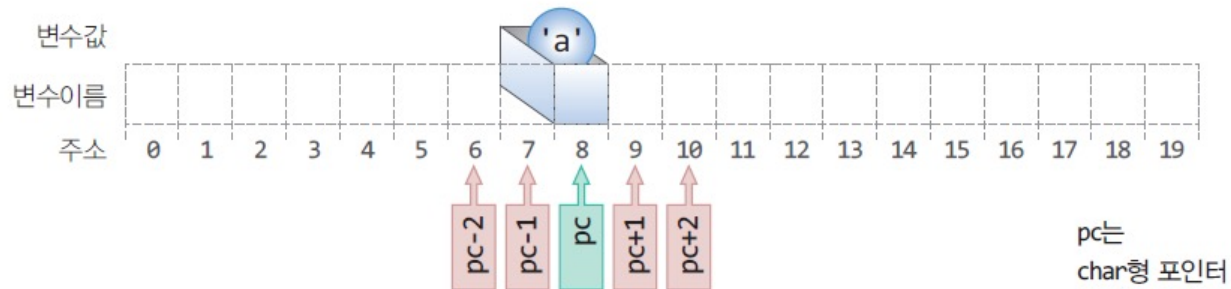
    printf("pc+2 = %d, pi+2 = %d, pd+2 = %d\n", pc+2, pi+2, pd+2);

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

```
증가 전 pc = 10000, pi = 10000, pd = 10000
증가 후 pc = 10001, pi = 10004, pd = 10008
```

# 포인터의 증감 연산



# 간접 참조 연산자와 증감 연산자

- `*p++;`
  - `p`가 가리키는 위치에서 값을 가져온 후에 `p`를 증가한다.
- `(*p)++;`
  - `p`가 가리키는 위치의 값을 증가한다.

수식	의미
<code>v = *p++</code>	<code>p</code> 가 가리키는 값을 <code>v</code> 에 대입한 후에 <code>p</code> 를 증가한다.
<code>v = (*p)++</code>	<code>p</code> 가 가리키는 값을 <code>v</code> 에 대입한 후에 가리키는 값을 증가한다.
<code>v = ++*p</code>	<code>p</code> 를 증가시킨 후에 <code>p</code> 가 가리키는 값을 <code>v</code> 에 대입한다.
<code>v = ++*p</code>	<code>p</code> 가 가리키는 값을 가져온 후에 그 값을 증가하여 <code>v</code> 에 대입한다.

```
1
2 #include<stdio.h>
```

```
3
4 int main(void)
```

```
{
```

```
5     int number = 10;
```

```
6     int *p;
```

```
7     int v = 0;
```

```
8
9
10    p = &number;
```

```
11
12    printf("number = %u \n", number);
```

```
13    printf("변수 number의 주소 = %u \n", &number);
```

```
14    printf("p = %u \n", p);
```

```
15
16    //v = *p++; // *p값을 v에 넣고, p는 증가 (int이므로 +4)
```

```
17    //v = (*p)++; // *p값을 v에 넣고, *p(=number값)를 ++
```

```
18    //v = *++p; //p를 증가시키고(int이므로 +4), 증가된 p가 가리키는 값을 v에 대입
```

```
19    v = ++ *p; // *p(=number값)를 가져와서, 그 값을 증가해서 v에 대입
```

```
20    printf("v = %u \n", v);
```

```
21    printf("p = %u \n", p);
```

```
22    printf("number = %u \n", number);
```

```
23
24    return 0;
```

```
25
26 }
```

number = 10  
변수 number의 주소 = 16383124  
p = 16383124  
v = 10  
p = 16383128  
number = 10

number = 10  
변수 number의 주소 = 5241872  
p = 5241872  
v = 10  
p = 5241872  
number = 11

number = 10  
변수 number의 주소 = 20379616  
p = 20379616  
v = 3435973836  
p = 20379620  
number = 10

number = 10  
변수 number의 주소 = 14155420  
p = 14155420  
v = 11  
p = 14155420  
number = 11

# 함수와 포인터

- 다른 사람에게 넘겨주어야 하는 정보가 상당히 방대하다고 하자. 이런 경우에는 전체를 복사해서 주는 것보다는 페이지 수만 알려주는 편이 간결할 수 있다.

정보들을 전부  
복사해 왔어요...

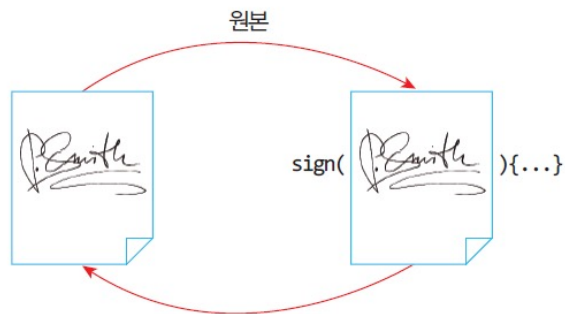
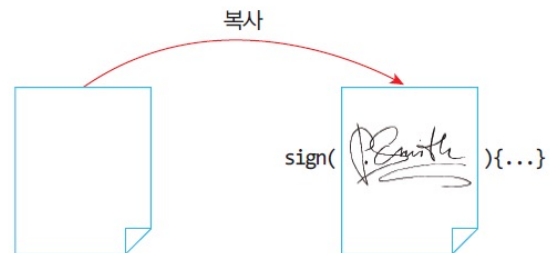


아이구  
그냥 p.300를 보라고  
말해주시자...  
다 복사해 오시다니



# 함수 호출시 인수 전달 방법

- 값에 의한 호출(call-by-value)
  - c의 기본적인 방법
  - 인수의 값이 매개 변수로 복사된다.
- 참조에 의한 호출(call-by-reference)
  - c에서는 포인터를 이용하여 흉내낼 수 있다.
  - 인수의 주소가 매개 변수로 복사된다.





# 값에 의한 호출

```
#include <stdio.h>

void modify(int value)
{
    value = 99;
}

int main(void)
{
    int number = 1;

    modify(number);
    printf("number = %d\n", number);

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

number = 1

# 참조에 의한 호출

```
#include <stdio.h>

void modify(int* ptr)
{
    *ptr = 99; // 매개 변수를 통하여 원본을 변경한다.
}

int main(void)
{
    int number = 1;

    modify(&number); // 주소를 계산해서 보낸다.
    printf("number = %d\n", number);

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

number = 99

# swap() 함수 #1

- 변수 2개의 값을 바꾸는 작업을 함수로 작성

```
int main(void)
{
    int a = 10, b = 20;

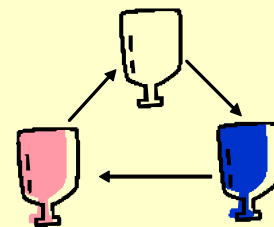
    swap(a, b);

    printf("swap() 호출 후 a=%d b=%d\n", a, b);
    return 0;
}
```

```
void swap(int x, int y)
```

```
{
    int tmp;

    tmp = x;
    x = y;
    y = tmp;
}
```



Microsoft Visual Studio 디버그 콘솔

swap() 호출 후 a=10 b=20

# swap() 함수 #2

- 포인터를 이용

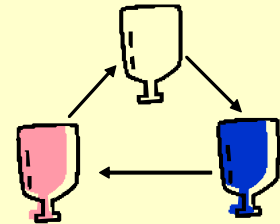
```
int main(void)
{
    int a = 100, b = 200;
    printf("swap() 호출 전 a=%d b=%d\n", a, b);

    swap(&a, &b);

    printf("swap() 호출 후 a=%d b=%d\n", a, b);
    return 0;
}
```

```
void swap(int *px, int *py)
{
    int tmp;

    tmp = *px;
    *px = *py;
    *py = tmp;
}
```



Microsoft Visual Studio 디버그 콘솔

```
swap() 호출전 a=100 b=200
swap() 호출후 a=200 b=100
```

# 참고사항



**Q** 값에 의한 호출과 참조에 의한 호출은 어떤 경우에 사용해야 하는가?

**A** 일반적으로 값에 의한 호출을 사용하여야 한다. 반면 함수가 외부에서 선언된 변수의 값을 변경할 필요가 있다면 포인터를 이용하여 "참조에 의한 호출" 효과를 낼 수 있다.

# 포인터 사용시 주의점

- 초기화가 안된 포인터를 사용하면 안된다.

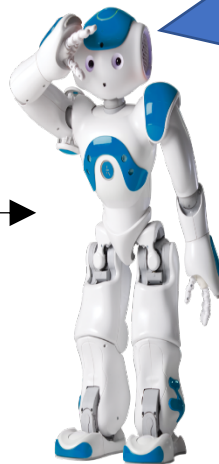
```
int main(void)
{
    int *p;           // 포인터 p는 초기화가 안되어 있음
    *p = 100;         // 위험한 코드
    return 0;
}
```



# 포인터 사용시 주의점

- 포인터가 아무것도 가리키고 있지 않는 경우에는 NULL로 초기화
- NULL 포인터를 가지고 간접 참조하면 하드웨어로 감지할 수 있다.
- 포인터의 유효성 여부 판단이 쉽다.

NULL 주소는 일반 사용자 금지구역입니다.



# 포인터 사용시 주의점

- 포인터의 타입과 변수의 타입은 일치하여야 한다.

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int i;
```

```
    double *pd;
```

```
    pd = &i;
```

```
    *pd = 36.5;
```

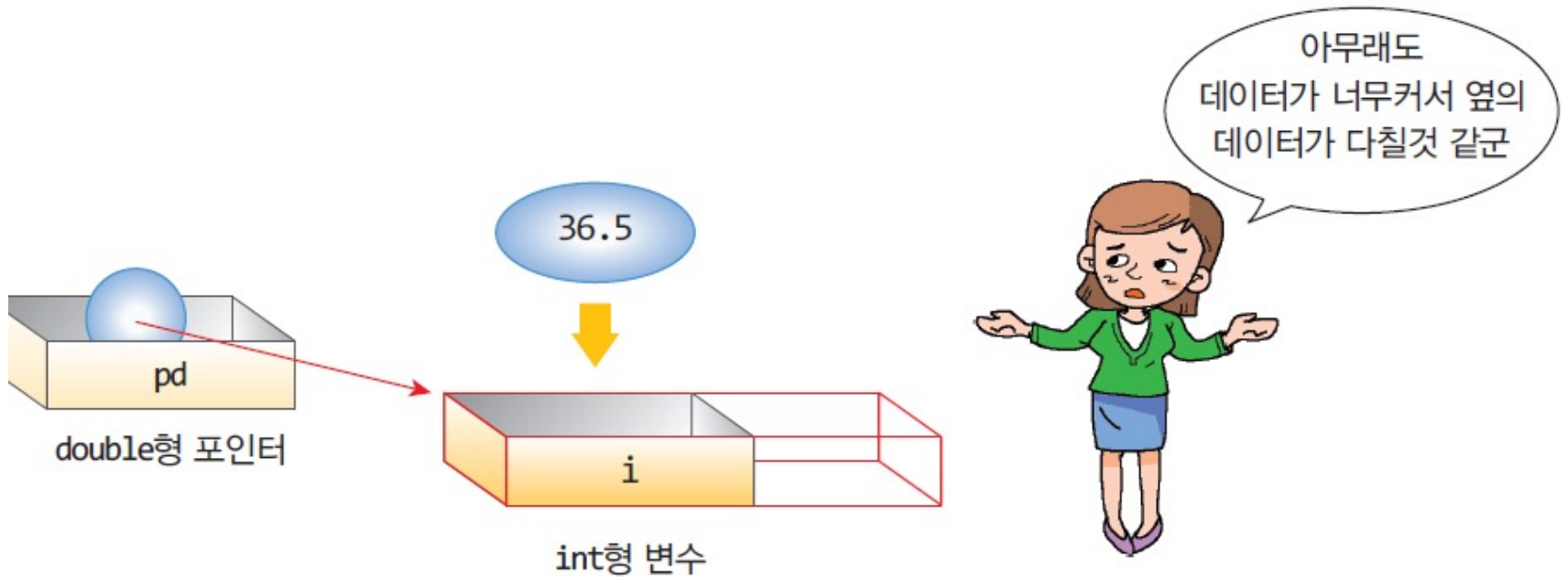
```
    return 0;
```

```
}
```

// 오류! double형 포인터에 int형 변수의 주소를 대입

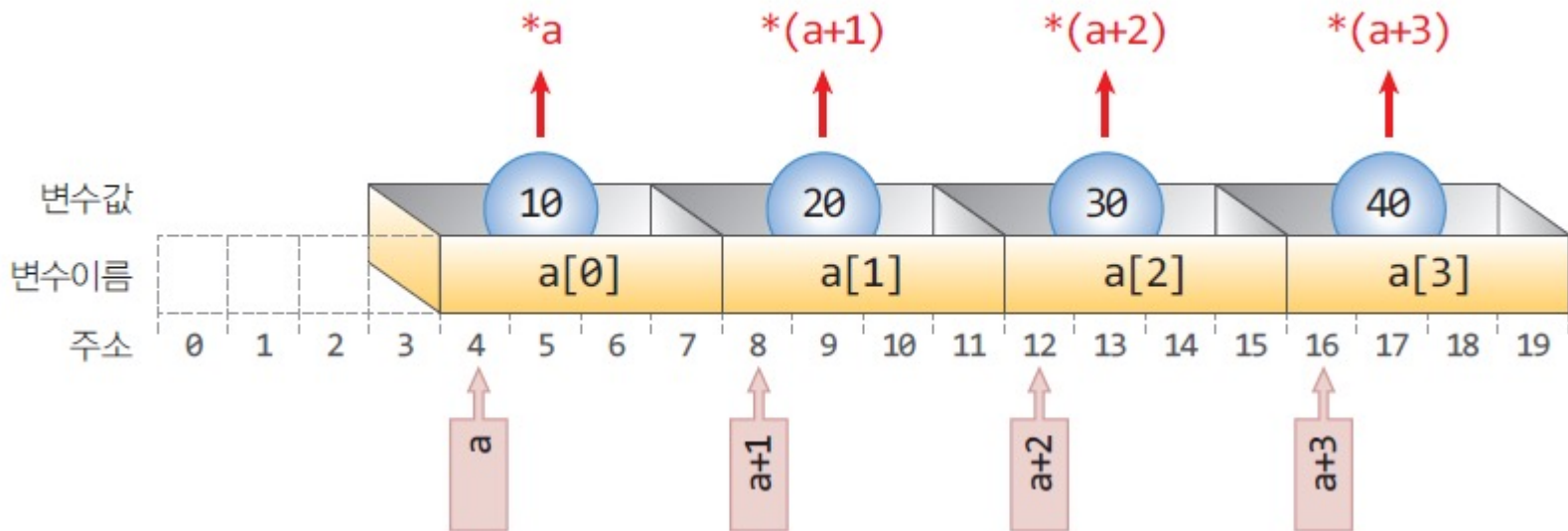


# 포인터 자료형과 변수의 자료형이 다른 경우



# 배열과 포인터

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



# 포인터와 배열

```
#include <stdio.h>

int main(void)
{
    int a[] = { 10, 20, 30, 40, 50 };

    printf("배열의 이름 = %u\n", a);
    printf("첫 번째 원소의 주소 = %u\n", &a[0]);

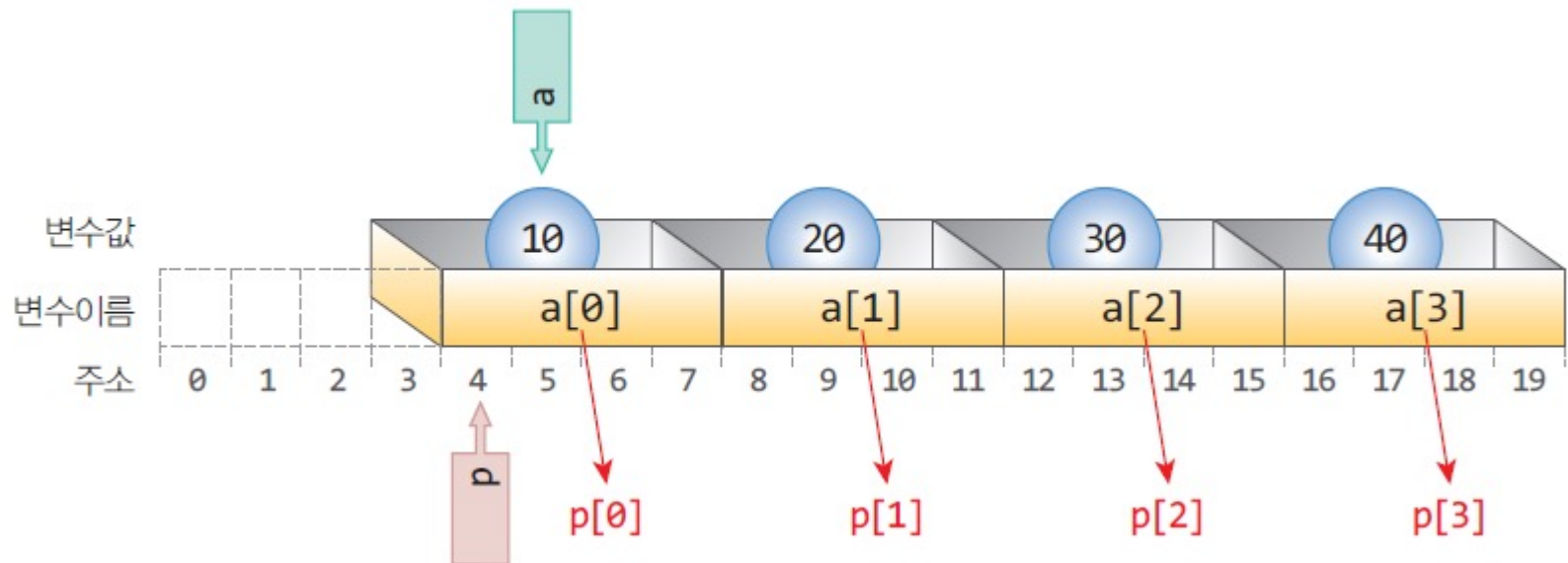
    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

배열의 이름 = 10877424  
첫 번째 원소의 주소 = 10877424

# 포인터를 배열처럼 사용

- 포인터도 배열이름처럼 간주될 수 있고 배열과 똑같이 사용할 수 있다.




# 포인터와 배열

```
#include <stdio.h>

int main(void)
{
    int a[] = { 10, 20, 30, 40, 50 };
    int* p;

    p = a;
    printf("a[0]=%d a[1]=%d a[2]=%d \n", a[0], a[1], a[2]);
    printf("p[0]=%d p[1]=%d p[2]=%d \n\n", p[0], p[1], p[2]);

    return 0;
}
```

 Microsoft Visual Studio 디버그 콘솔

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

# 중간 점검



## 중간점검

1. 배열 `a[]`에서 `*a`의 의미는 무엇인가?
2. 배열의 이름에 다른 변수의 주소를 대입할 수 있는가?
3. 포인터를 이용하여 배열의 원소들을 참조할 수 있는가?
4. 포인터를 배열의 이름처럼 사용할 수 있는가?

# 어디에 사용될까?

- 대용량 데이터를 모두 복사해서 주는 대신 그 데이터가 있는 위치를 알려줘!

```
#include <stdio.h>
void sub(int* ptr)
{
    printf("%d \n", ptr[10]);
}

int main(void)
{
    int large_data[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 };

    sub(large_data);
    return 0;
}
```

# 중간점검



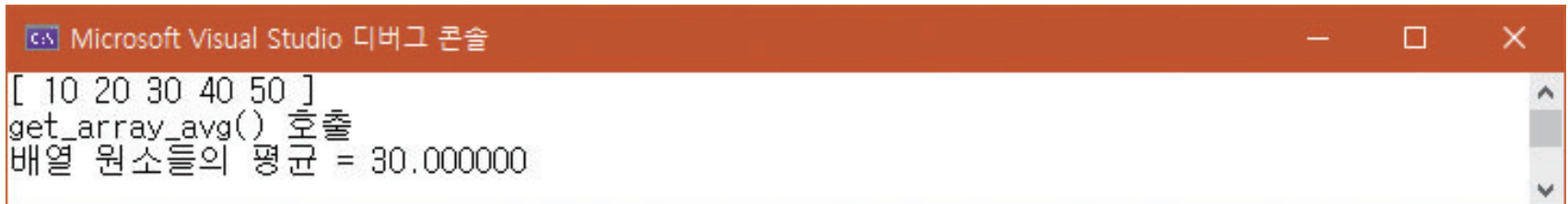
## 중간점검

1. 함수에 매개 변수로 변수의 복사본이 전달되는 것을 \_\_\_\_\_ 라고 한다.
2. 함수에 매개 변수로 변수의 원본이 전달되는 것을 \_\_\_\_\_ 라고 한다.
3. 배열을 함수의 매개 변수로 지정하는 경우, 배열의 복사가 일어나는가?



# Lab: 유용한 배열 함수 작성

- 정수 배열에 대하여 평균을 계산하고 배열을 출력하는 함수를 작성하고 사용해보자.
  - `double get_array_avg(int values[], int n);` 정수 배열을 받아서 배열 요소의 평균값을 계산하여 반환한다.
  - `void print_array(int values[], int n);` 정수 배열을 받아서 배열 요소들을 출력한다.



```
C:\> Microsoft Visual Studio 디버그 콘솔
[ 10 20 30 40 50 ]
get_array_avg() 호출
배열 원소들의 평균 = 30.000000
```

The screenshot shows the Microsoft Visual Studio Debug Console window. The title bar is orange and contains the text 'C:\> Microsoft Visual Studio 디버그 콘솔' along with standard window control buttons (minimize, maximize, close). The console area has a white background and displays the following text: '[ 10 20 30 40 50 ]', 'get\_array\_avg() 호출', and '배열 원소들의 평균 = 30.000000'. A vertical scrollbar is visible on the right side of the console area.

Sol:

```
#include <stdio.h>
#define SIZE 5
double get_array_avg(int values[], int n);
void print_array(int values[], int n);

int main(void)
{
    int i;
    int data[SIZE] = { 10, 20, 30, 40, 50 };
    double result;

    print_array(data, SIZE);
    result = get_array_avg(data, SIZE);
    printf("배열 원소들의 평균 = %f\n", result);
    return 0;
}
```

// 배열 요소의 평균을 계산하는 함수

double get\_array\_avg(int values[], int n)

```
{  
    int i;  
    double sum = 0.0;  
    for (i = 0; i < n; i++)  
        sum += values[i];  
    return sum / n;  
}
```

// 배열 요소를 화면에 출력하는 함수

void print\_array(int values[], int n)

```
{  
    int i;  
    printf("[ ");  
    for (i = 0; i < n; i++)  
        printf("%d ", values[i]);  
    printf("]\n");  
}
```

# Mini Project: 어드벤처 게임 만들기

- 간단한 텍스트 기반의 게임을 작성해보자. 주인공은 '#'로 표시되어 있다. 주인공이 금 'G'를 찾으면 게임이 종료된다.
- 중간에 몬스터 'M'가 있어서 금을 찾는 것을 방해한다.
- 주인공은 'w', 's', 'a', 'd' 키를 이용하여 상하좌우로 움직일 수 있다.
- 몬스터는 랜덤하게 움직이는 것으로 하라.

