

# **Basic Computer Programming**

## Lecture 9

Electrical & Electronics Engineering  
Chung-Ang University

# Contents

- Understand the concept of pointers.
- Understand the pointer declaration and initialization process.
- Understand the specificity of the operation of the pointer.
- Understand the relationship between the pointer and the array.
- Understand the call by reference using a pointer.

# The program we will make in this chapter

- Let's write a program that calculates the address of a variable.

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

- Let's write a function swap() that replaces the contents of variables a and b.

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

- Let's define and use functions that process an array.

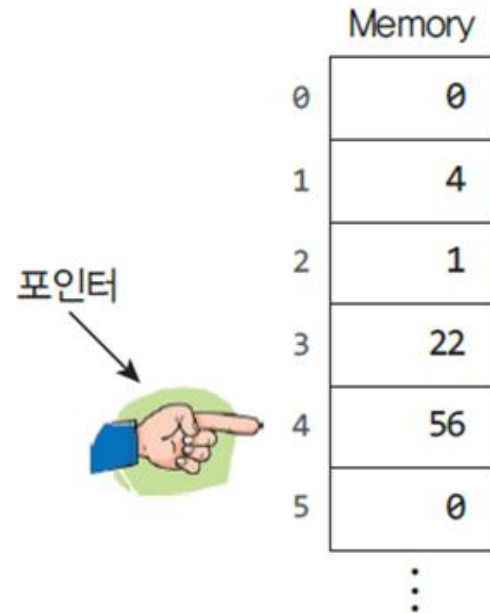
```
Microsoft Visual Studio 디버그 콘솔
[ 10 20 30 40 50 ]
get_array_avg() 호출
배열 원소들의 평균 = 30.000000
```

# Pointer?

- *Pointer*: variables with address

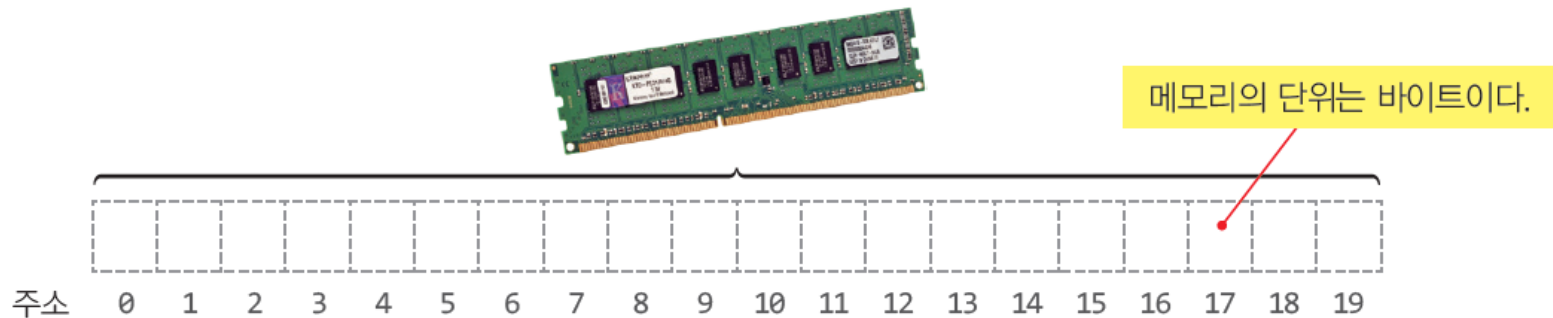


집



# How Memory Works

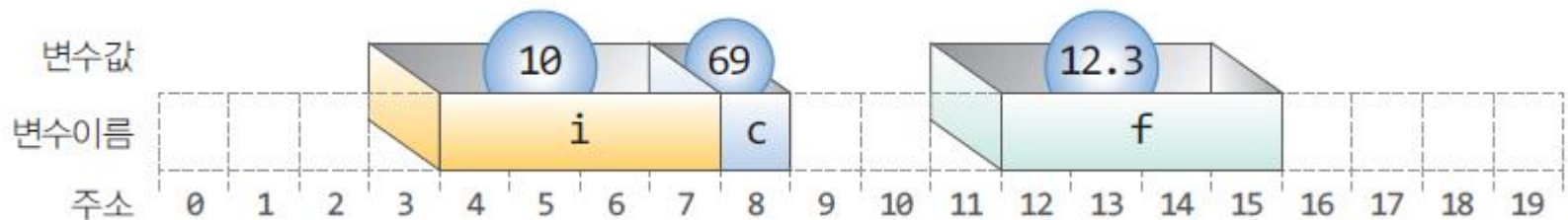
- The variable is stored in the memory.
- The memory is accessed in bytes.
- The address of the first byte is 0, and the second byte is 1,...



# Variables and Memory

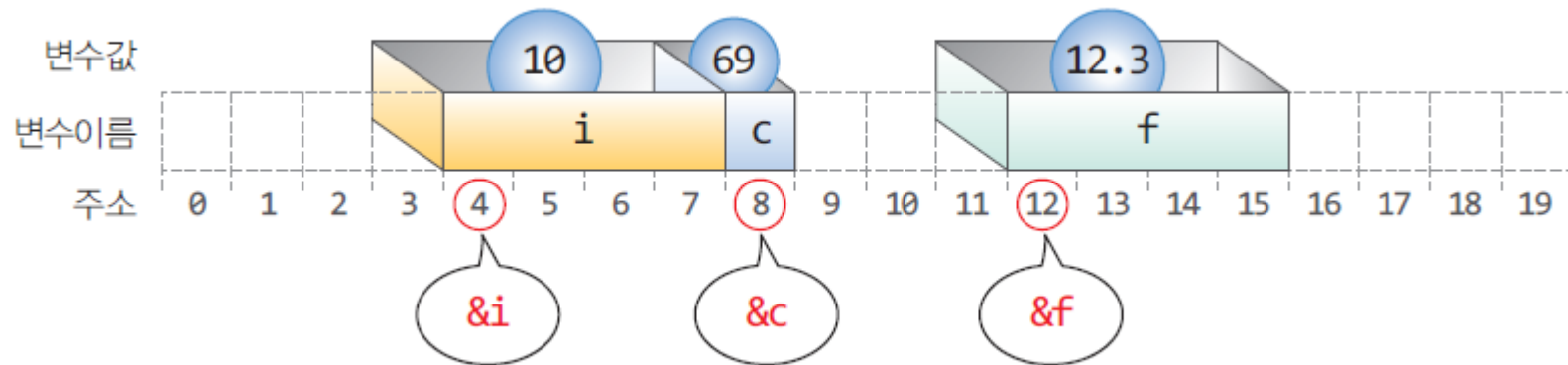
- The amount of memory occupied depends on the size of the variable.
- char type variable: 1 byte, int type variable: 4 bytes,...

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



# Address of variable

- Operator that calculates the address of a variable: &
- Address of variable i: &i



# Address of variable

```
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

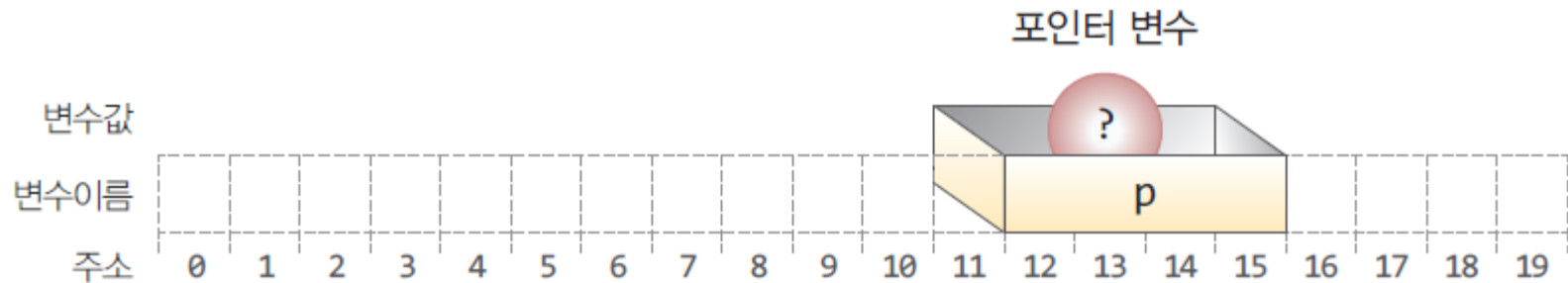
```
i의 주소: 012FFA0C
c의 주소: 012FFA03
f의 주소: 012FF9F0
```



# Declaration of Pointers

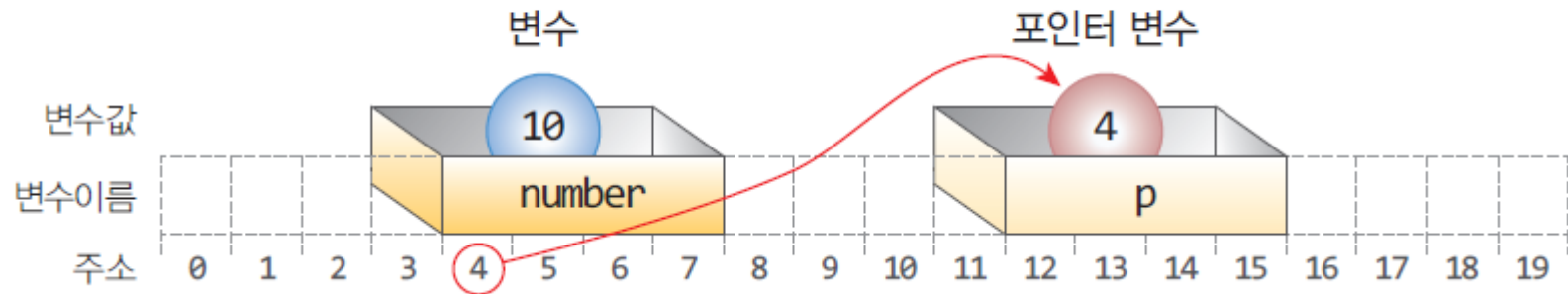
- Pointers: Variables with addresses of variables

```
int *p
```



# Associating Pointers with Variables

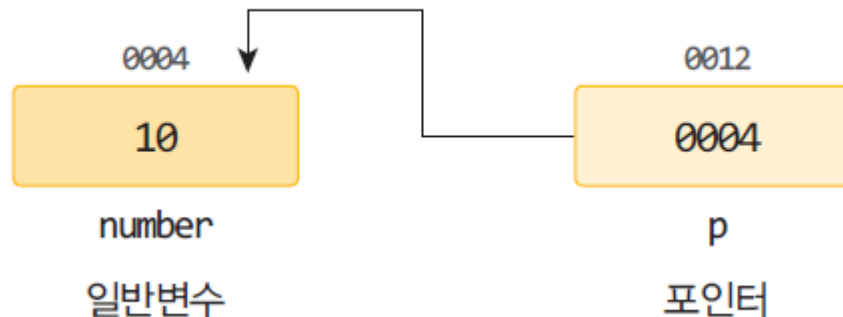
```
int    number = 10;           // Integer variable number declaration
int    *p;                    // Declare pointer variable p
p = &number;                  // Assign the address to pointer p
```



# Pointers and Variables

- The pointer p indicates the variable number.

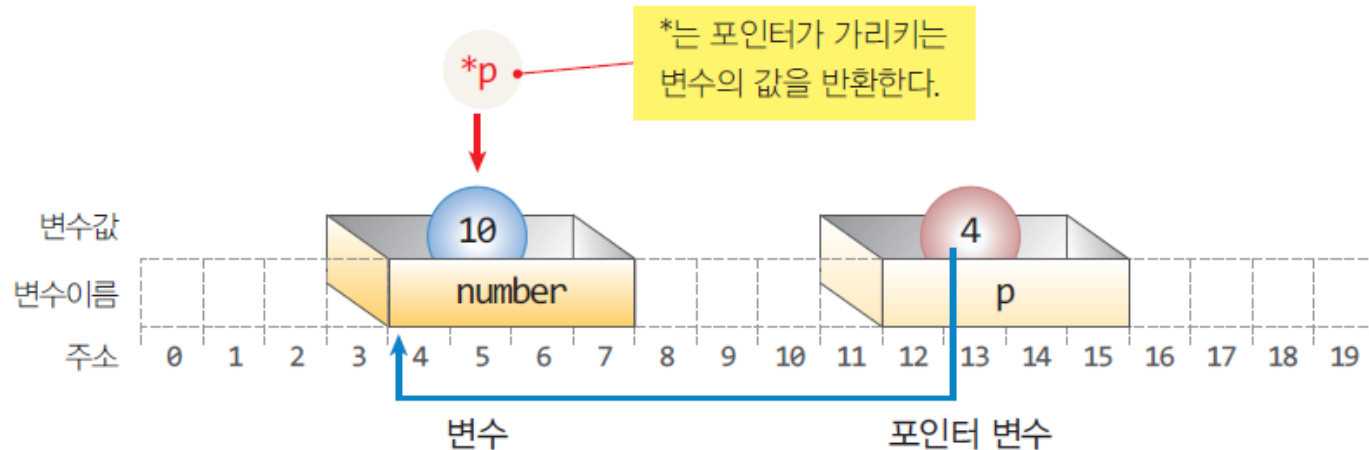
```
int    number = 10;    // 정수형 변수 number 선언
int    *p;             // 포인터 변수 p 선언
p = &number;           // 변수 number의 주소가 포인터 p로 대입
```



# Indirect Reference Operator

- Indirect Reference Operator \*: The operator that gets the value that the pointer points to

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



# Pointer operators

- There are two operators related to pointers:



# Example #1

```
#include <stdio.h>

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

    p = &number;

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

# Example #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

# Interim check



## 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.





# Pointer operation

- Possible operations: Increasing, decreasing, adding, subtracting operations
- For incremental operations, the increasing value is the size of the object that the pointer points to

$++p;$

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

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



# Pointer operation example

```
#include <stdio.h>
int main(void)
{
    char *pc;
    int *pi;
    double *pd;

    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);

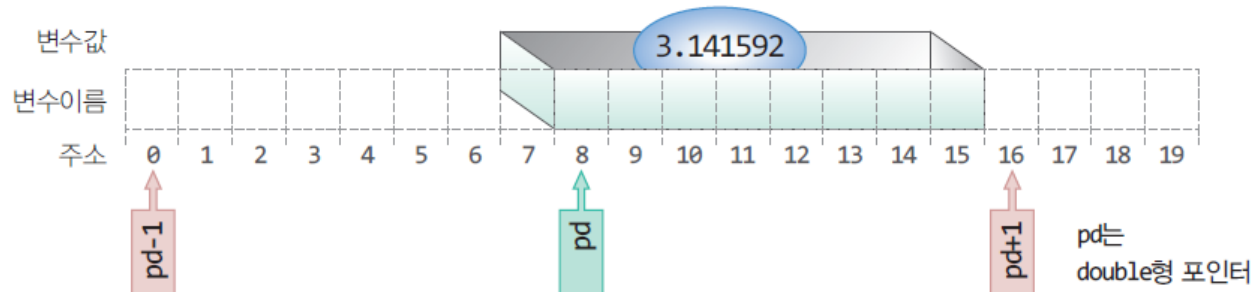
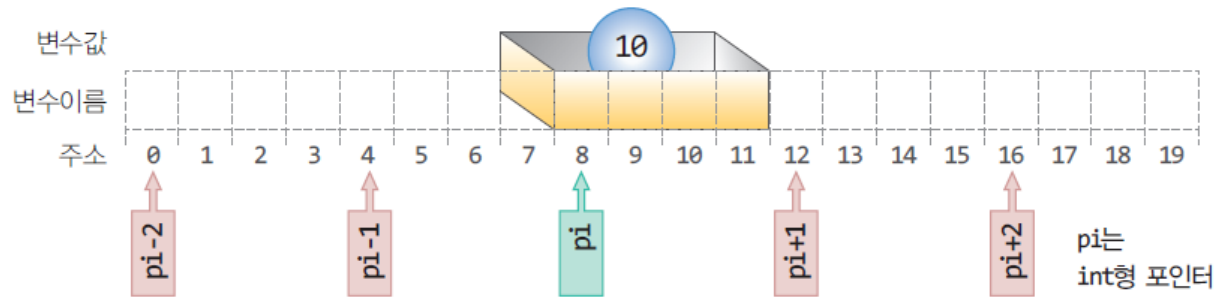
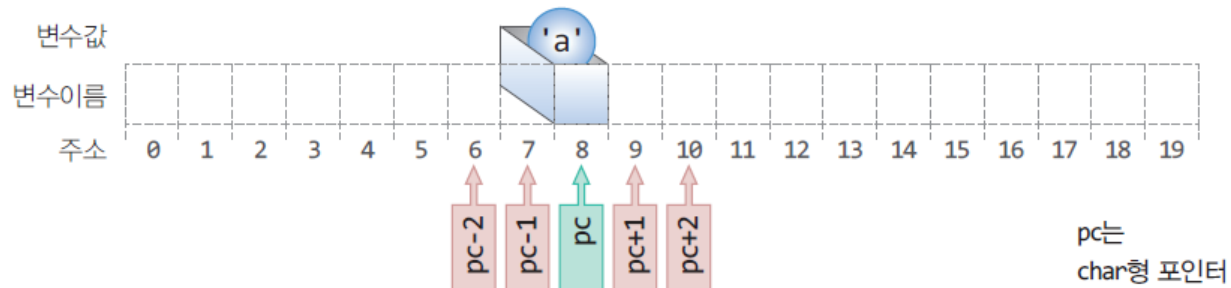
    printf("pc+2 = %d, pi+2 = %d, pd+2 = %d\n", pc+2, pi+2, pd+2);

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

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

# Increasing and decreasing operations for pointers



# Indirect Reference Operator and Incremental Operator

- $*p++$ ;
  - After the value is taken from the position where  $p$  is pointed,  $p$  is increased.
- $(*p)++$ ;
  - Increase the value of the position where  $p$  points.

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

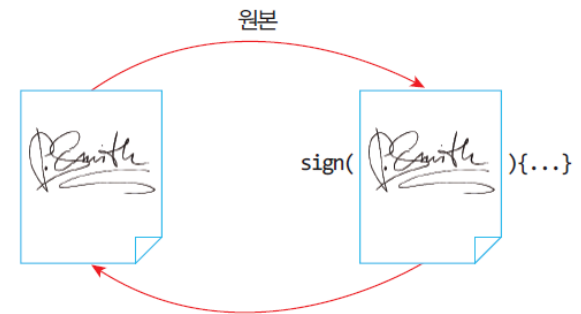
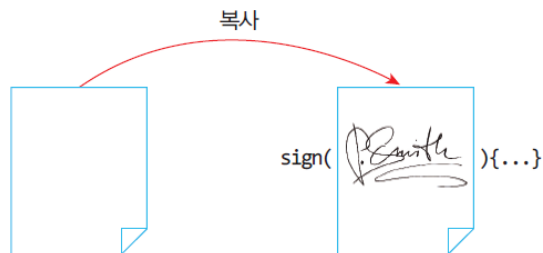
# Functions and Pointers

- Let's say that the information that needs to be handed over to others is quite vast. In this case, it may be simpler to tell only the page rather than to copy the whole thing.



# Providing arguments when calling a function

- Call-by-value
  - the basic method of C
  - The value of the argument is copied as a parameter.
- Call-by-reference
  - In C, you can imitate it using a pointer.
  - The address of the argument is copied as a parameter.



# Call-by-value

```
#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

# Call-by-reference

```
#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() function #1

- Write a function to change the values of two variables

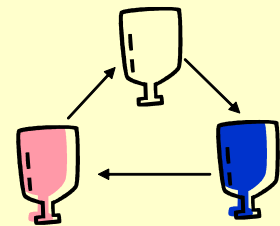
```
int main(void)
{
    int a = 10, b = 20;

    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() function #2

- Using pointers

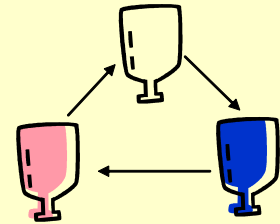
```
int main(void)
{
    int a = 100, b = 200;
    printf("swap() 호출 전 a=%d b=%d\n", a, b);

    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
```

# Check



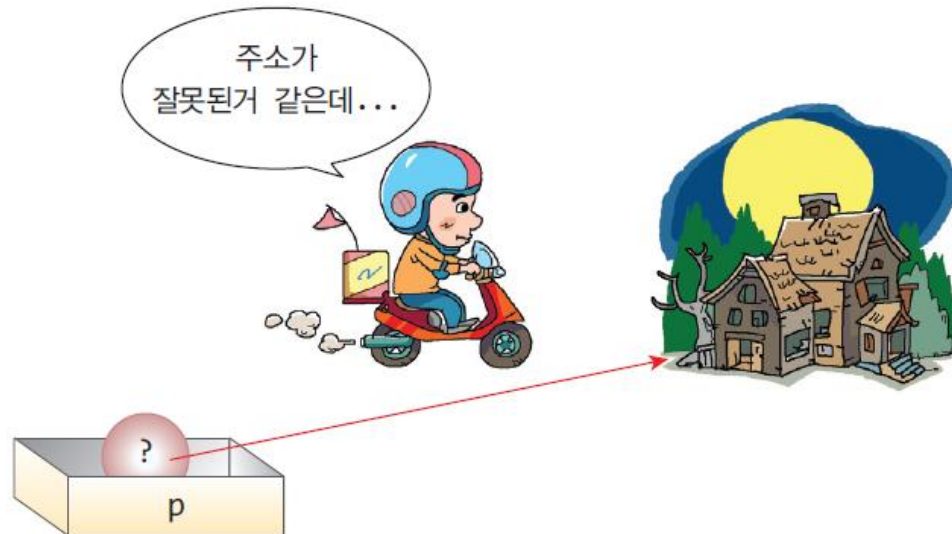
**Q** 값에 의한 호출과 참조에 의한 호출은 어떤 경우에 사용해야 하는가?

**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.

# Cautions for using pointers

- Do not use uninitialized pointers.

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



# Cautions for using pointers

- The type of pointer and the type of variable must match.

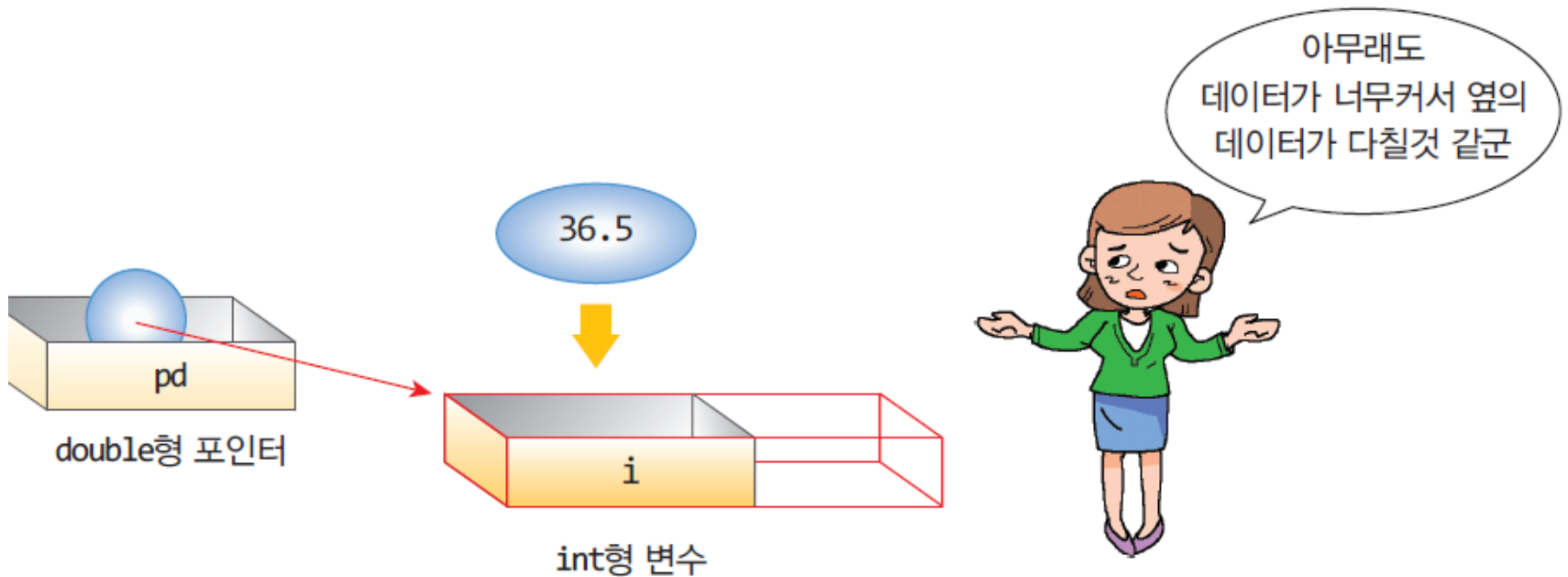
```
#include <stdio.h>

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

    pd = &i // Error! assigning address of int type variable in double type pointer
    *pd = 36.5;

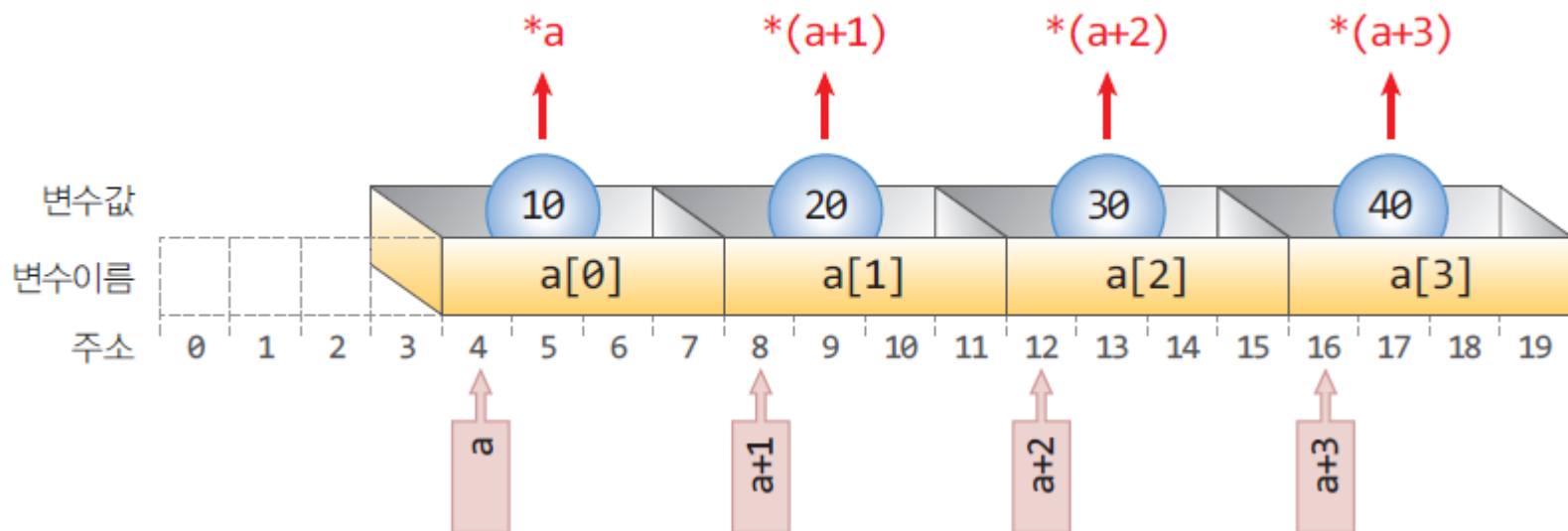
    return 0;
}
```

# If pointer data type and variable data type are different



# Arrays and Pointers

- The array and the pointer have a very close relationship.
- The name of the array is the pointer.
- Pointers can be used like arrays.



# Pointers and Arrays

```
#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;
}
```

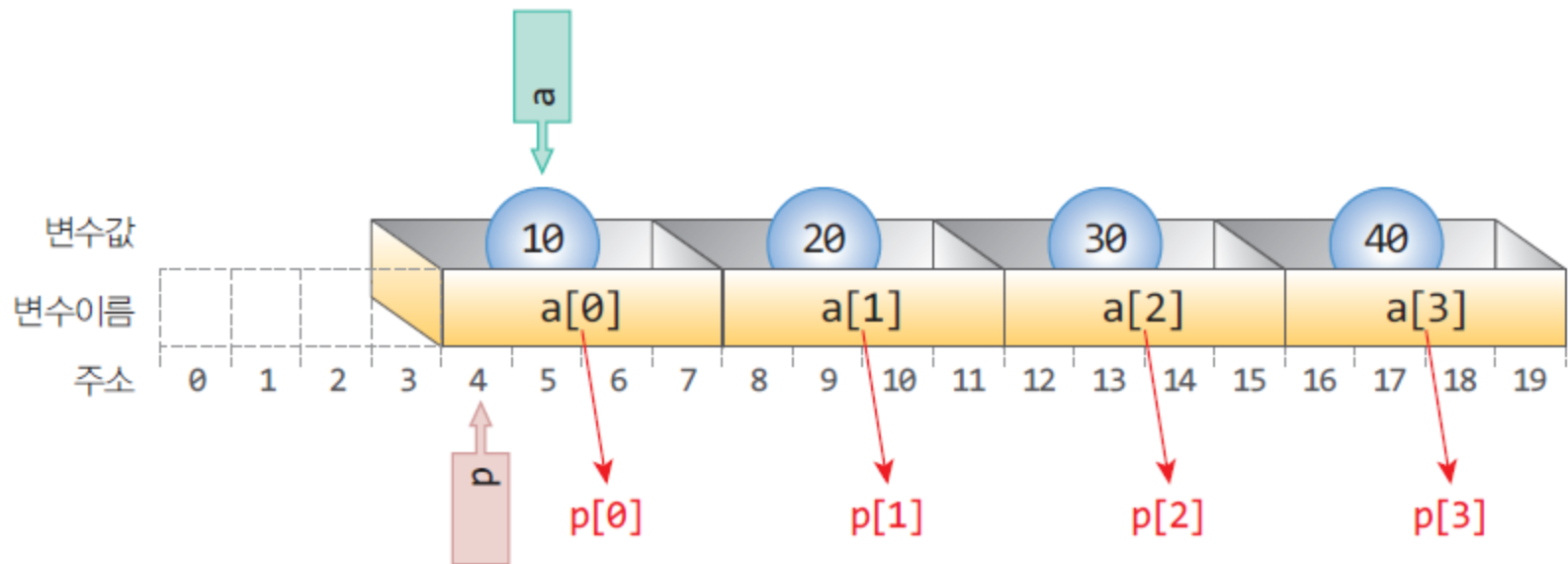
Microsoft Visual Studio 디버그 콘솔

배열의 이름 = 10877424  
첫 번째 원소의 주소 = 10877424



# Using pointers as an Array

- Pointers can also be considered as array names and can be used in the same way as arrays.



# Pointers and Arrays

```
#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
```

# Interim check



## 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?

# Where will it be used?

```
#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;
}
```

Microsoft Visual Studio 디버그 콘솔

11

# Interim check

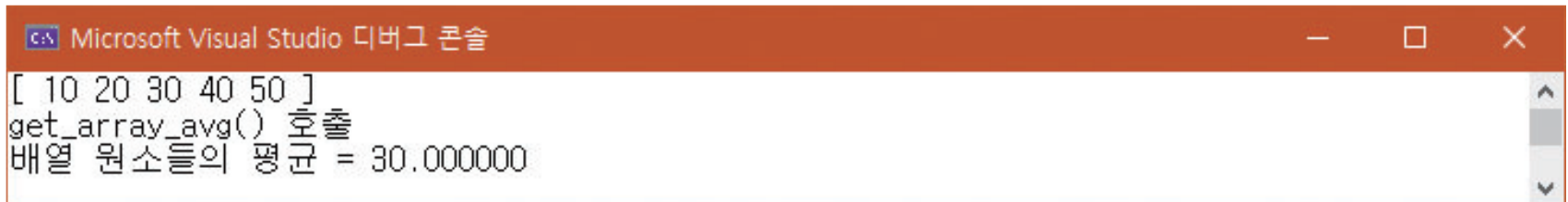


## 중간점검

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: Create useful array functions

- Let's write and use a function that averages and outputs an array for an integer array.
- `double get_array_avg(int values[], int n);` takes an integer array and calculates and returns the average value of the array element.
- `void print_array(int values[], int n);` receives an integer array and outputs array elements.



The screenshot shows the 'Microsoft Visual Studio 디버그 콘솔' (Debug Console) window. It contains the following text: an array `[ 10 20 30 40 50 ]`, the text `get_array_avg() 호출` (calling get\_array\_avg()), and the output `배열 원소들의 평균 = 30.000000` (Average of array elements = 30.000000). The window has a standard Windows title bar with minimize, maximize, and close buttons.

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

# 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;
}
```

# Sol.

// 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");
}
```