



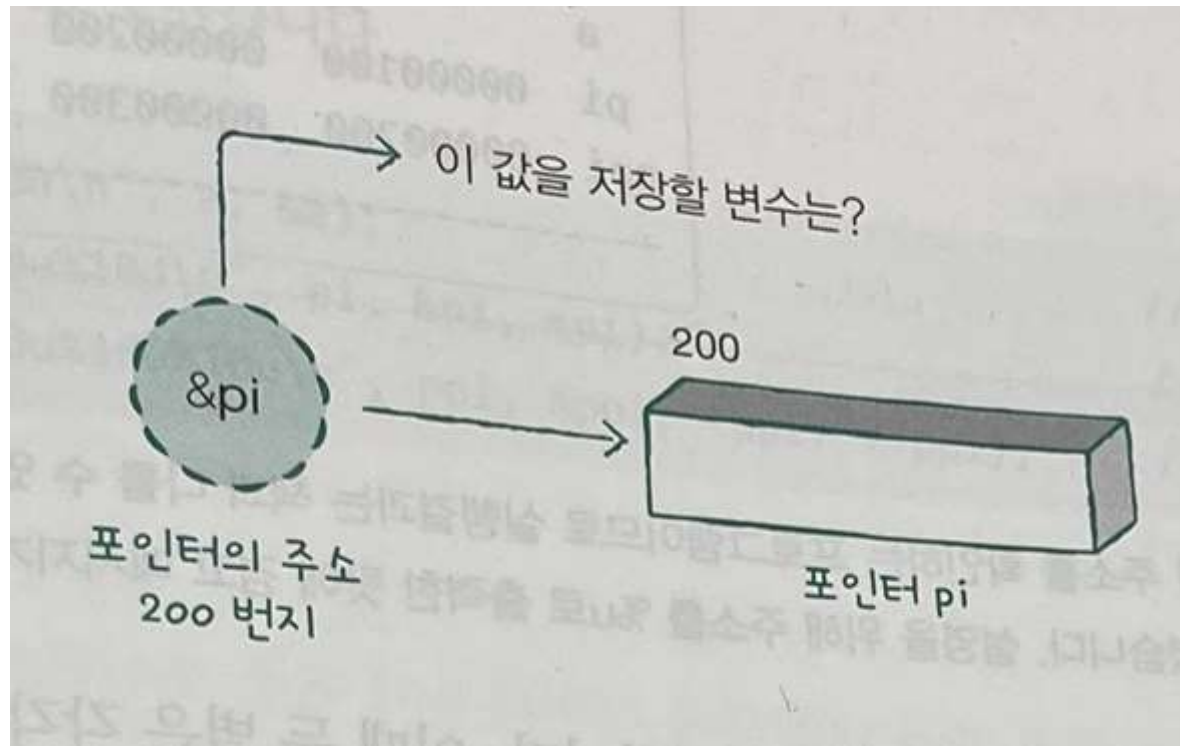
혼.공.C Chapter15

2021210088 허지혜

이중 포인터란?

포인터 = 변수의 주소 값

이중 포인터 - 포인터 변수의 주소 값



이중 포인터 예시

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

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

```
    int a = 10;
```

```
    int* pi;
```

```
    int** ppi;
```

```
    pi = &a;
```

```
    ppi = &pi;
```

```
    printf("-----\n");
```

```
    printf("변수 변수값 &연산 *연산 **연산\n");
```

```
    printf("-----\n");
```

```
    printf("a%10d%10u\n", a, &a);
```

```
    printf("pi%10u%10u%10d\n", pi, &pi, *pi);
```

```
    printf("ppi%10u%10u%10u%10u\n", ppi, &ppi, *ppi, **ppi);
```

```
    printf("-----\n");
```

```
    return 0;
```

```
}
```

첫번째 *: ppi가 가리키는 자료형이 포인터
두번째 *: ppi 자신이 포인터

Microsoft Visual Studio 디버그 콘솔

```
변수 변수값 &연산 *연산 **연산
```

```
a      10 11532336
```

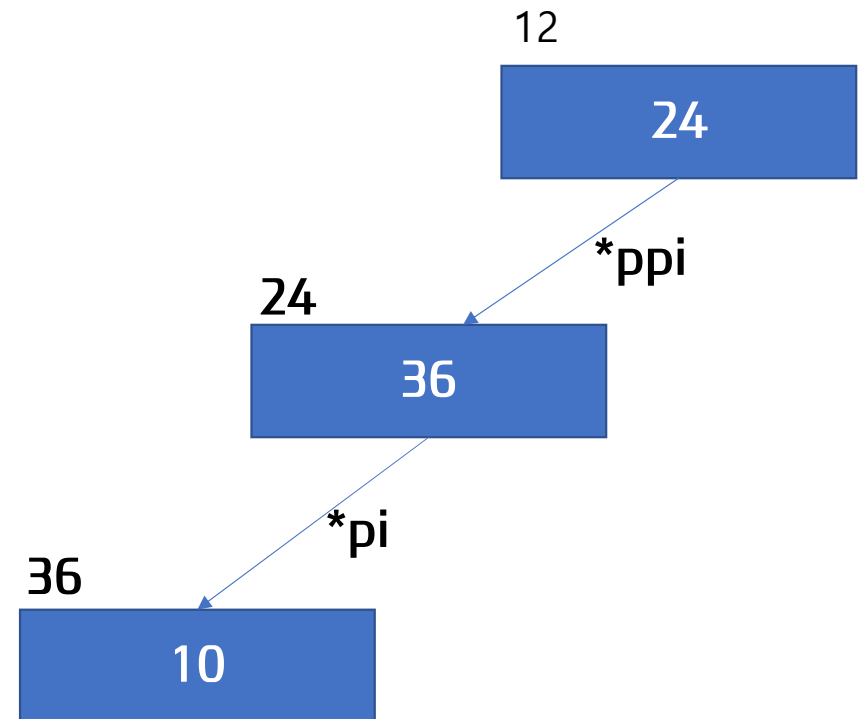
```
pi 11532336 11532324      10
```

```
ppi 11532324 11532312 11532336      10
```

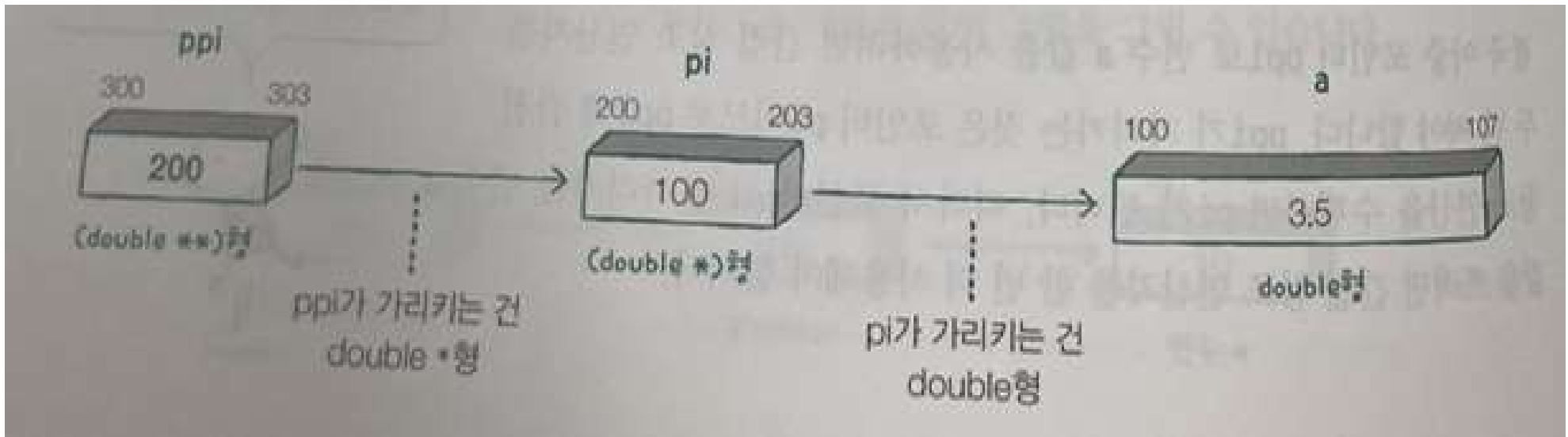
C:\Users\huh612\source\repos\Project2\Debug\Proj
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도
하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...

이중 포인터 예시

변수	변수 값	&연산	*연산	**연산
a	10	36		
pi	36	24	10	
ppi	24	12	36	10



이중 포인터 형태



- 포인터가 가리키는 것의 형태와 포인터 자신의 형태를 구분해야 한다.
- 이중 포인터 `ppi`를 지정하면 `ppi`가 가리키는 자료형은 `(double *)`형이고 `pi`가 가리키는 자료형은 `double`형이다.
- 이때 `double`은 포인터가 가리키는 자료형에 대한 정보지 포인터 자체를 의미하는 것이 아니라서 `pi`와 `ppi`의 byte는 4byte이다.

다중 포인터

이중 포인터
Double **ppi

삼중 포인터(다중 포인터)
Double ***ppp

이런 방식으로 진행하는데 가독성을 떨어뜨리기 때문에 이중 이상의 포인터를 다중 포인터라고 부른다.

이중 포인터 활용 : 포인터 값을 바꾸는 매개변수

```
#include <stdio.h>

void swap_ptr(char** ppa, char** ppb);

int main(void)
{
    char* pa = "success";
    char* pb = "failure";

    printf("pa->%s, pb->%s\n", pa, pb);
    swap_ptr(&pa, &pb);
    printf("pa->%s, pb->%s\n", pa, pb);

    return 0;
}

void swap_ptr(char** ppa, char** ppb)
{
    char* pt;
    pt = *ppa;
    *ppa = *ppb;
    *ppb = pt;
}
```

Microsoft Visual Studio 디버깅

```
pa->success, pb->failure
pa->failure, pb->success

C:\Users\whu612\source\repos
디버깅이 중지될 때 콘솔을 개
하도록 설정합니다.
이 창을 닫으려면 아무 키나
```

이중 포인터 활용 : 포인터 배열을 매개변수로 받는 함수

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

```
void print_str(char** pps, int cnt);
```

```
int main(void)
```

```
{
```

```
    char* ptr_ary[] = { "eagle", "tiger", "lion", "squirrel" };
```

```
    int count;
```

```
    count = sizeof(ptr_ary) / sizeof(ptr_ary[0]);
```

```
    print_str(ptr_ary, count);
```

```
    return 0;
```

```
}
```

```
void print_str(char** pps, int cnt)
```

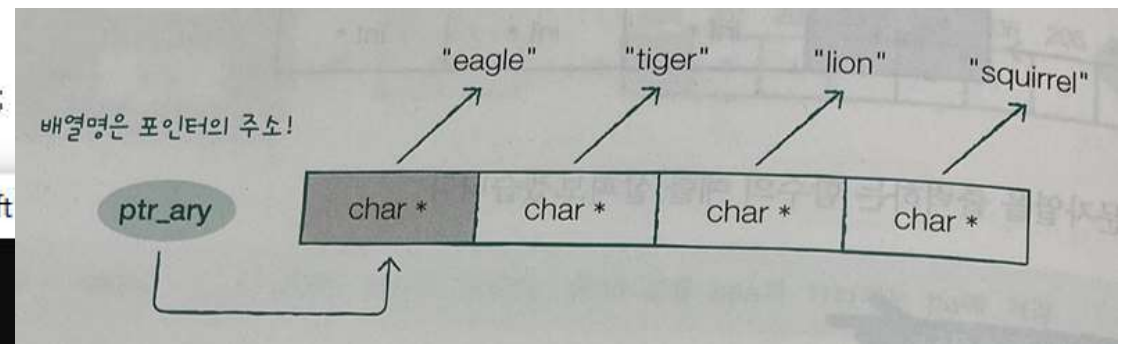
```
{
```

```
    int i;
```

```
    for (i = 0; i < cnt; i++)
```

```
    {
```

```
        printf("%s\n", pps[i]);
```



Microsoft

```
eagle
tiger
lion
squirrel
```

```
C:\Users\hu612\source\rep
디버깅이 중지될 때 콘솔을
하도록 설정합니다.
이 창을 닫으려면 아무 키
```


배열 요소의 주소와 배열의 주소

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

```
int main(void)
```

```
{
```

```
    int ary[5];
```

```
    printf("ary값 : %u\n", ary);
```

```
    printf("ary의 주소 : %u\n", &ary);
```

```
    printf("ary+1 : %u\n", ary + 1);
```

```
    printf("&ary+1 : %u\n", &ary + 1);
```

```
    return 0;
```

```
}
```

ary 와 &ary는 주소로 쓰일 때와 주소 연산자를 사용한
모습이므로 주소는 동일하다.

하지만 ary + 1은 두번째 요소를 가리키므로 +4의 모습
이고 &ary + 1은 배열 전체를 가리키므로 + 20이 된다.

Microsoft Visual Studio 디버그 콘솔

```
ary값 : 4521612 ary의 주소 : 4521612  
ary+1 : 4521616  
&ary+1 : 4521632
```

2차원 배열과 배열 포인터

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

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

```
    int ary[3][4] = { {1,2,3,4}, {5,6,7,8}, {9,10,11,12} };
```

```
    int(*pa)[4];
```

```
    int i, j;
```

```
    pa = ary;
```

```
    for (i = 0; i < 3; i++)
```

```
    {
```

```
        for (j = 0; j < 4; j++)
```

```
        {
```

```
            printf("%5d", pa[i][j]);
```

```
        }
```

```
        printf("\n");
```

```
    }
```

```
    return 0;
```

```
}
```

가리키는 것은 int 4개의 1차원 배열이고 pa는 포인터이다.

포인터에 배열명을 저장하면 배열처럼 쓸 수 있다.

Microsoft Visual Studio 디버그 콘솔

```
1   2   3   4
5   6   7   8
9  10  11  12
```

C:\Users\whu612\source\repos\Project2\De
디버깅이 중지될 때 콘솔을 자동으로 닫으

2차원 배열과 배열 포인터

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

```
void print_ary(int(*)[4]);
```

```
int main(void)
```

 $\{$

```
int ary[3][4] = { {1,2,3,4}, {5,6,7,8}, {9,10,11,12} };
```

```
print_ary(ary);
```

```
return 0;
```

}

```
void print_ary(int(*pa)[4])
```

 $\{$

```
int i, j;
```

```
for (i = 0; i < 3; i++)
```

10

```
for (j = 0; j < 4; j++)
```

11

```
printf("%5d", pa[i][j]);
```

}

```
printf("%d\n");
```

1

Microsoft Visual Studio

1	2	3	4
5	6	7	8
9	10	11	12

C:\Users\hu612\source\tr
디버깅이 중지될 때 콘솔
하도록 설정합니다.
이 창을 닫으려면 아무 키

2차원 배열 요소의 두가지 의미

2차원 배열은 논리적으로 1차원의 부분 배열을 뜻하고 물리적으로는 실제 데이터를 저장하는 부분 배열의 요소를 뜻한다.

```
int ary[3][4];
```

2차원 배열 ary의 논리적 배열 요소의 개수는? 3개

2차원 배열 ary의 물리적 배열 요소의 개수는? 12개

2차원 배열의 요소를 참조하는 원리

```
Int ary[3][4];
```

12개의 배열 요소에서 7번째 요소를 꺼내는 원리 = `ary[1][2]`

1. `ary + 1` -> `100 + (1 * sizeof(ary[0]))` -> `100 + (1*16)` -> 116

`*(ary+1) = ary[1]` 이므로 여기에 2를 더하면 된다.

2. `*(ary+1)+2` -> `*(ary+1)+(2*sizeof(ary[1][0]))` -> `116 + (2*4)` -> 124

`*(*(ary+1)+2)` -> `ary[1][2]` //두 번째 부분배열의 세 번째 배열 요소

마무리

- ✓ 포인터도 하나의 변수이므로 그 주소가 있다.
- ✓ 이중 포인터에 간접 참조 연산자 *를 사용하면 단일 포인터가 된다.
- ✓ 2차원 배열의 배열명은 첫 번째 부분 배열의 주소가 된다.
- ✓ 배열 포인터에 간접 참조 연산자를 사용하면 가리키는 배열이 된다.

구분	기능	설명
이중 포인터	선언 방법	<code>int **p;</code>
	사용 예1	포인터를 교환하는 함수의 매개변수로 사용
	사용 예2	포인터 배열을 처리하는 함수의 매개변수로 사용
배열 포인터	선언 방법	<code>int (*pa)[4];</code>
	사용 예1	int형 변수 4개짜리 1차원 배열을 가리킨다.
	사용 예2	2차원 배열의 배열명을 받는 함수의 매개변수에 사용

확인 문제

1. 다음과 같이 변수가 선언되고 그림처럼 메모리에 할당되어 있을 때 출력문의 결과를 적어 보세요. 100, 200, 300은 각 변수의 주소입니다.

```
double grade = 4.5;
double *pg = &grade;
double **ppg = &pg;

printf("%.1lf\n", **ppg);
printf("%u\n", &ppg);
printf("%u\n", *&pg);
printf("%u\n", *ppg);
printf("%u\n", &*ppg);
```

4.5
300
100
100
200

확인 문제

2. 다음 프로그램의 실행결과를 예상해보세요.

```
#include <stdio.h>

int main(void)
{
    int a = 10, b = 20;
    int* pa = &a, * pb = &b;
    int** ppa = &pa, ** ppb = &pb;
    int* pt;

    pt = *ppa;
    *ppa = *ppb;
    *ppb = pt;

    printf("a:%d, b:%d\n", a, b);
    printf("*pa:%d, *pb:%d\n", *pa, *pb);

    return 0;
}
```

A:10 b:20
*pa:20 *pb:10

함수 포인터와 void 포인터

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

```
int sum(int, int);
```

```
int main(void)
```

```
{  
    int (*fp)(int, int);  
    int res;
```

함수의 주소 sum을 저장할 함수 포인터 선언
가리키는 함수의 형태를 반환값과 매개 변수로 나누어 적는다.

```
    fp = sum;
```

```
    res = fp(10, 20);
```

```
    printf("result : %d\n", res);
```

```
    return 0;
```

```
}
```

```
int sum(int a, int b)
```

```
{
```

```
    return (a + b);
```

```
}
```

Microsoft Visu

result : 30

C:\Users\whu612\디버깅이 중지될
하도록 설정합니
이 창을 닫으려

함수 포인터의 활용

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

```
void func(int (*fp)(int, int));
int sum(int a, int b);
int mul(int a, int b);
int max(int a, int b);
```

```
int main(void)
{
    int sel;

    printf("01 두 정수의 합\n");
    printf("02 두 정수의 곱\n");
    printf("03 두 정수 중에서 큰 값 계산\n");
    printf("원하는 연산을 선택하세요 : ");
    scanf("%d", &sel);

    switch (sel)
    {
        case 1: func(sum); break;
        case 2: func(mul); break;
        case 3: func(max); break;
    }
}
```

```
void func(int (*fp)(int, int))
{
    int a, b;
    int res;

    printf("두 정수의 값을 입력하세요 : ");
    scanf("%d%d", &a, &b);
    res = fp(a, b);
    printf("결과값은 : %d\n", res);
}
```

```
int sum(int a, int b)
{
    return (a + b);
}
```

```
int mul(int a, int b)
{
    return (a * b);
}
```

```
int max(int a, int b)
{
    if (a > b) return a;
    else return b;
}
```

```
01 두 정수의 합
02 두 정수의 곱
03 두 정수 중에서 큰 값 계산
원하는 연산을 선택하세요 : 2
두 정수의 값을 입력하세요 : 3 7
결과값은 : 21
```

Void 포인터

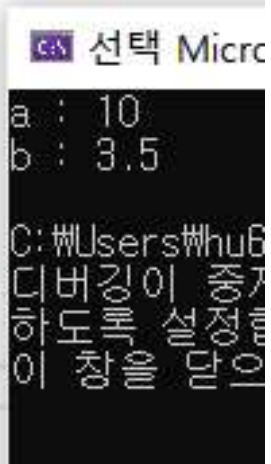
```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
    int a = 10;
    double b = 3.5;
    void* vp;

    vp = &a;
    printf("a : %d\n", *(int*)vp);

    vp = &b;
    printf("b : %.1lf\n", *(double*)vp);

    return 0;
}
```



```
선택 Micro
a : 10
b : 3.5
C:\Users\huB
디버깅이 중지
하도록 설정함
이 창을 닫으
```

주소를 가리키는 자료형이 일치할 경우 포인터에 대입이 가능하다.

Void 포인터는 가리키는 자료형이 정해지지 않은 포인터이다. 따라서 어떤 주소든 저장할 수 있다.

또한 같은 이유로 간접 참조 연산이나 정수를 더하는 포인터 연산이 불가능하다.

사용하려면 원하는 형태로 변환하여 사용하면 된다.

마무리

- ✓ 함수명의 의미부터 보자면 함수명은 함수 정의가 있는 메모리의 시작 주소이다.
- ✓ 함수 포인터에 함수명을 대입하면 함수처럼 호출할 수 있다.
- ✓ void 포인터에는 임의의 주소를 저장할 수 있다.
- ✓ void 포인터는 간접 참조 연산과 주소에 대한 정수 연산이 불가능하다.

구분	기능	설명
함수 포인터	선언 방법	<code>int (*fp)(int, int);</code>
	함수의 호출	<code>fp(10,20);</code>
	용도	함수명을 대입하여 호출 함수를 결정한다.
void 포인터	선언 방법	<code>void *vp;</code>
	의미	가리키는 자료형에 대한 정보가 없다.
	용도	임의의 주소를 받는 함수의 매개변수에 사용한다.

확인 문제

1. 다음과 같이 선언된 함수가 있을 때 대입 연산이 가능하도록 함수 포인터를 선언 하세요.

```
double div(int, int);  
void prn(char *);  
int *save(int);
```

①

②

③

```
fpa = div;           // ①의 대입 연산  
fpb = prn;           // ②의 대입 연산  
fpc = save;          // ③의 대입 연산
```

Double (*fpa)(int, int);
Char (*fpb)(char);
Int (*fpc)(int);

확인 문제

2. 다음의 배열과 포인터가 있을 때 포인터 vp로 세 번째 배열 요소의 값 30을 출력하세요.

```
#include <stdio.h>

int main(void)
{
    int ary[5] = { 10, 20, 30, 40, 50 };
    void *vp = ary;

    printf(    );

    return 0;
}
```

답

확인 문제

3. 다음 프로그램의 실행결과를 예상해보세요.

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

```
int add(int a, int b) { return (a + b); }  
int sub(int a, int b) { return (a - b); }  
int mul(int a, int b) { return (a * b); }
```

```
int main(void)  
{  
    int (*pary[3])(int, int) = { add, sub, mul };  
    int i, res = 0;  
  
    for (i = 0; i < 3; i++)  
    {  
        res += pary[i](2, 1);  
    }  
    printf("%d", res);  
  
    return 0;  
}
```

6

끝