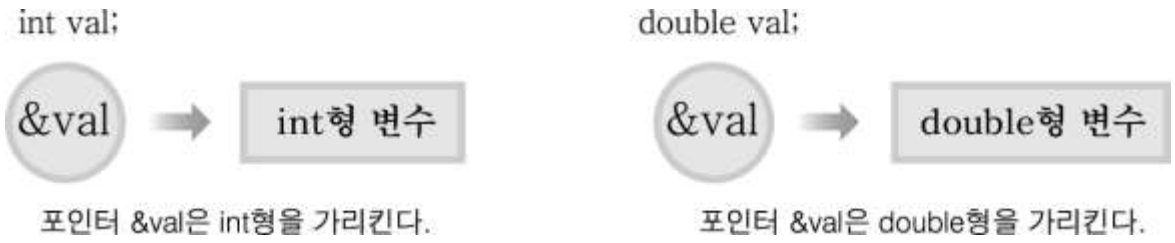
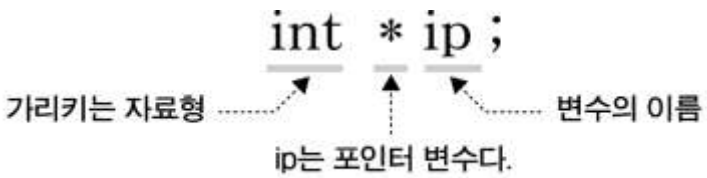
응용 포인터

❑ 포인터의 핵심 정리

- 포인터는 다음의 두 가지만 이해하면 응용이 가능하다!
  - 첫째, 포인터가 가리키는 자료형은 무엇인가?



  - 둘째, 포인터는 같은 자료형을 가리키는 포인터변수에 저장해야 한다.

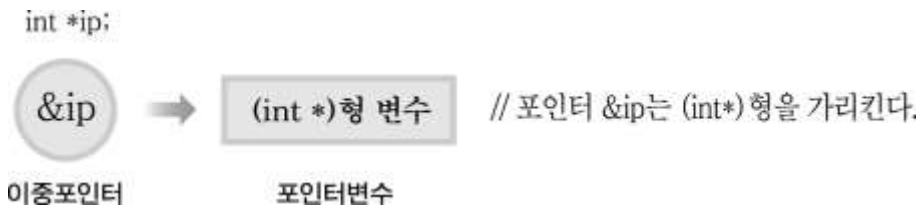



```
ip = &val ;   // 포인터를 포인터변수에 대입
```

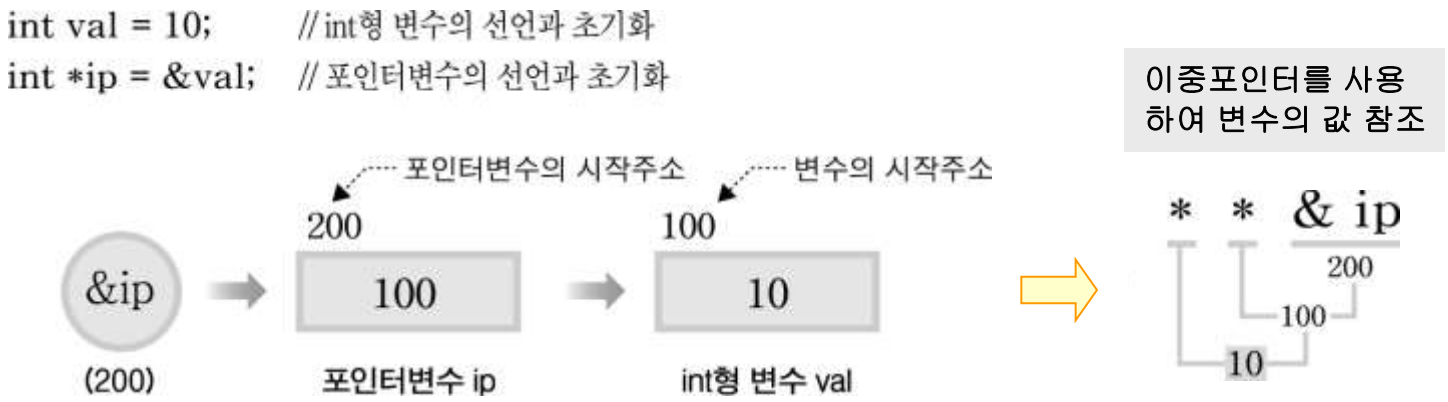
가리키는 자료형은 모두 int형으로 같다!

❑ 다중 포인터

- 포인터변수도 하나의 기억공간이므로 포인터를 구할 수 있다. 포인터변수의 포인터가 이중포인터이다.

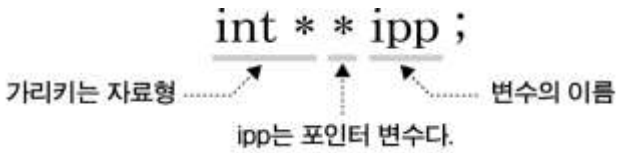


- 이중포인터로 변수를 참조할 때는 참조연산자를 두 번 사용해야 한다.



▶ 이중포인터변수

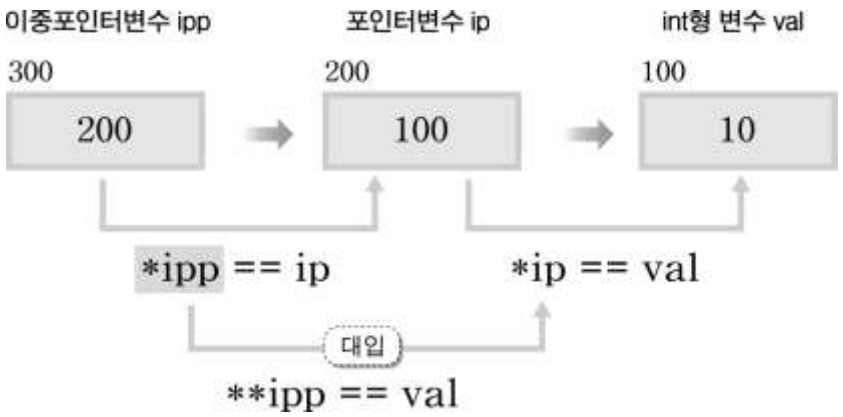
- 이중포인터 또한 하나의 포인터이므로 포인터변수에 저장하여 사용한다.




```
ipp = &ip ;  // 이중포인터를 이중포인터변수에 대입
```
가리키는 자료형은 모두 (int *)형으로 같다!

- 이중포인터변수도 참조연산자를 두 번 사용하여 변수를 참조한다.

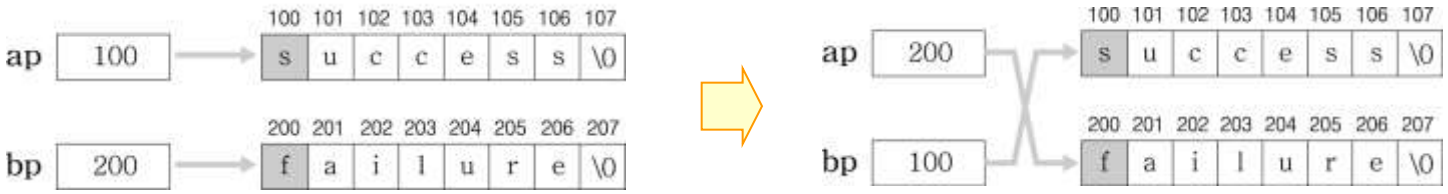
```
int val=10;
int *ip;      // 포인터변수
int **ipp;    // 이중포인터변수

ip=&val;
ipp=&ip;
printf("%d₩n", **ipp);
```



▶ 이중포인터변수의 사용 예

- 두 포인터변수의 값을 바꾸는 함수를 만들어보자.

```
char *ap = “success”;
char *bp = “failure”;
```

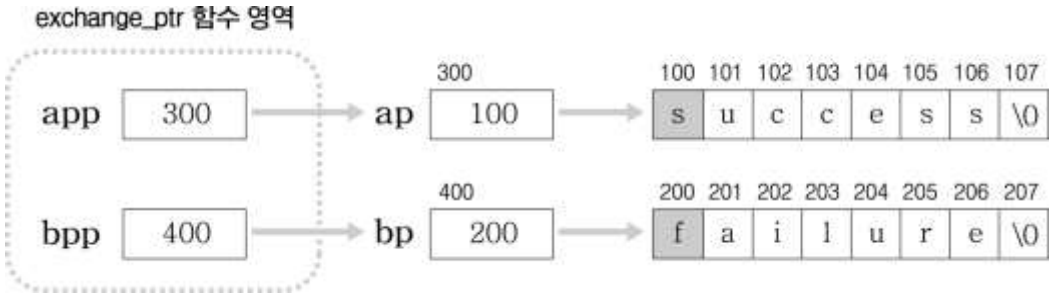


함수의 호출 `exchange_ptr(&ap, &bp);` // 이중포인터를 전달인자로 준다.

함수의 선언 `void exchange_ptr(char **app, char **bpp);`

// 이중포인터가 전달인자로 넘어오므로 매개변수로 이중포인터변수를 선언한다.

함수가 호출된 후

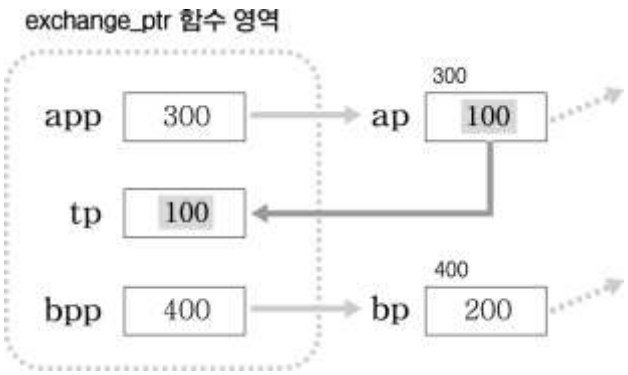


▶ 이중포인터변수의 사용 예

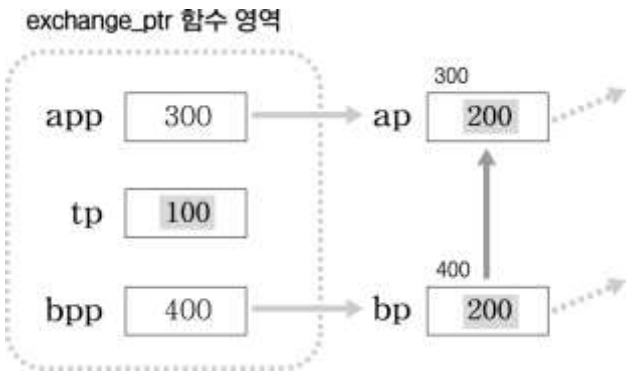
- 함수 안에서는 임시 포인터변수를 선언하여 ap, bp의 값을 바꾼다.

```
char *tp;    // 임시 포인터변수의 선언
```

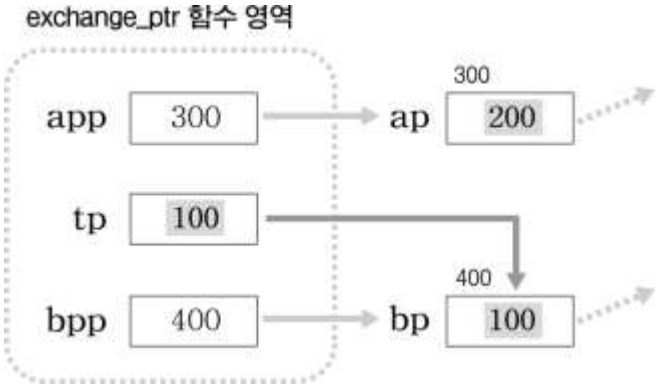
① tp = *app;
// app가 가리키는 곳에
// 저장된 값을 tp에 저장한다.



② *app = *bpp;
// bpp가 가리키는 곳에 저장된 값을
// app가 가리키는 곳에 저장한다.



③ *bpp = tp;
// tp의 값을 bpp가 가리키는
// 곳에 저장한다.



▶ 함수를 사용하여 포인터변수의 값을 바꾸는 프로그램

```
#include <stdio.h>
void exchange_ptr(char **, char **);                    // 함수의 선언
int main()
{
    char *ap="success";
    char *bp="failure";
    printf("ap -> %s, bp -> %s₩n", ap, bp);             // 바꾸기 전의 문자열 출력
    exchange_ptr(&ap, &bp);                             // 함수의 호출
    printf("ap -> %s, bp -> %s₩n", ap, bp);             // 바꾼 후에 문자열 출력
    return 0;
}

void exchange_ptr(char **app, char **bpp)               // 함수의 정의
{
    char *tp;                                           // 임시 포인터변수
    tp=*app;            // 메인함수에 있는
    *app=*bpp;          // ap, bp의 값을 바꾼다.
    *bpp=tp;
}
```

출력 결과

```
ap -> success, bp -> failure
ap -> failure, bp -> success
```

❑ 배열 포인터

- 배열 포인터는 배열 전체를 가리키는 포인터다.
- 2차원 배열을 처리하는 함수의 매개변수로는 배열포인터를 쓴다.
  - 2차원 배열 ary의 모든 값을 출력하는 함수를 만들 때

```
int ary[3][4] = { {1, 2, 3, 4}, {11, 12, 13, 14}, {21, 22, 23, 24} };
```

함수의 호출

```
ary_prn(ary);   // 배열명을 전달인자로 주고 호출한다.
```

함수의 선언

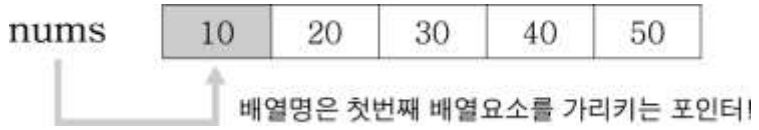
```
void ary_prn( int (*ap)[4] );
```

매개변수로 배열포인터가 필요하다!

▶ 배열명의 정체

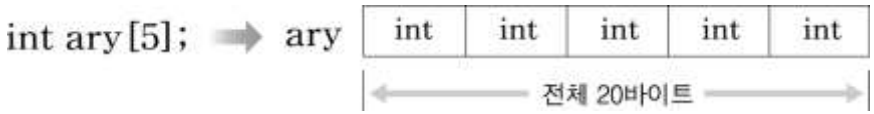
- 배열명은 첫 번째 배열요소를 가리키는 포인터의 기능을 한다.

```
int nums[5]={10, 20, 30, 40, 50};
int *ip=nums;            // 배열명은 포인터이므로 포인터변수에 저장한다.
printf("%d₩n", *nums);          // 첫번째 기억공간을 참조하여 10이 출력된다.
printf("%d₩n", *(nums+4));      // nums의 값이 1000이면 116번지를 참조한다.
```



- 배열의 기억공간 전체를 나타내는 논리적 변수의 기능을 한다.

  – 논리적 변수이므로 직접 데이터를 저장할 수는 없으나 응용 자료형으로서 변수가 가지는 형태와 크기에 대한 정보를 가진다.



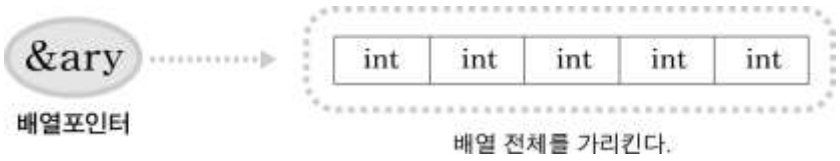
크기 ⇨ 20바이트

형태 ⇨ int형 변수 다섯 개의 배열형

▶ 배열포인터가 가리키는 것은?

- 배열포인터는 배열명에 주소연산을 수행하면 구해진다.
  - 이 때 배열명은 논리적변수의 기능이며 그 배열 전체를 가리키는 포인터가 구해진다.

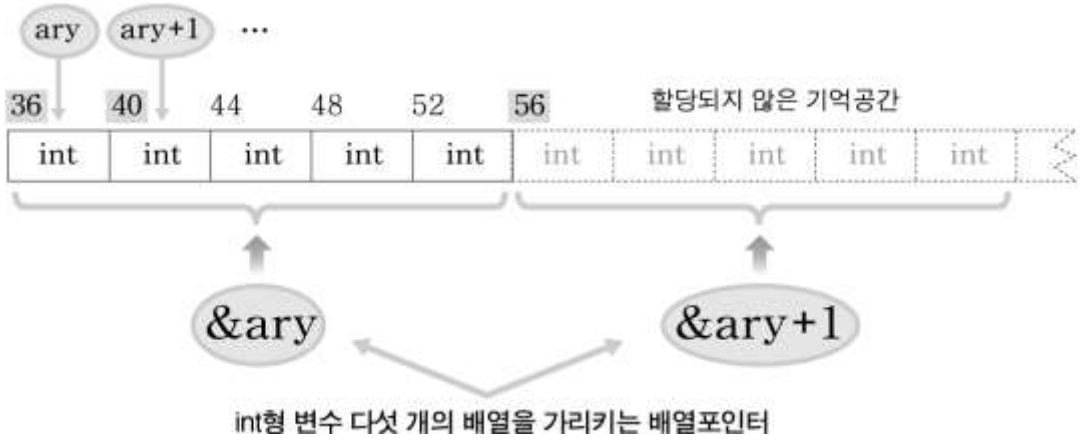
int ary[5];



  - 배열포인터에 정수값을 더하면 배열 전체의 크기를 곱해서 더해주게 된다.

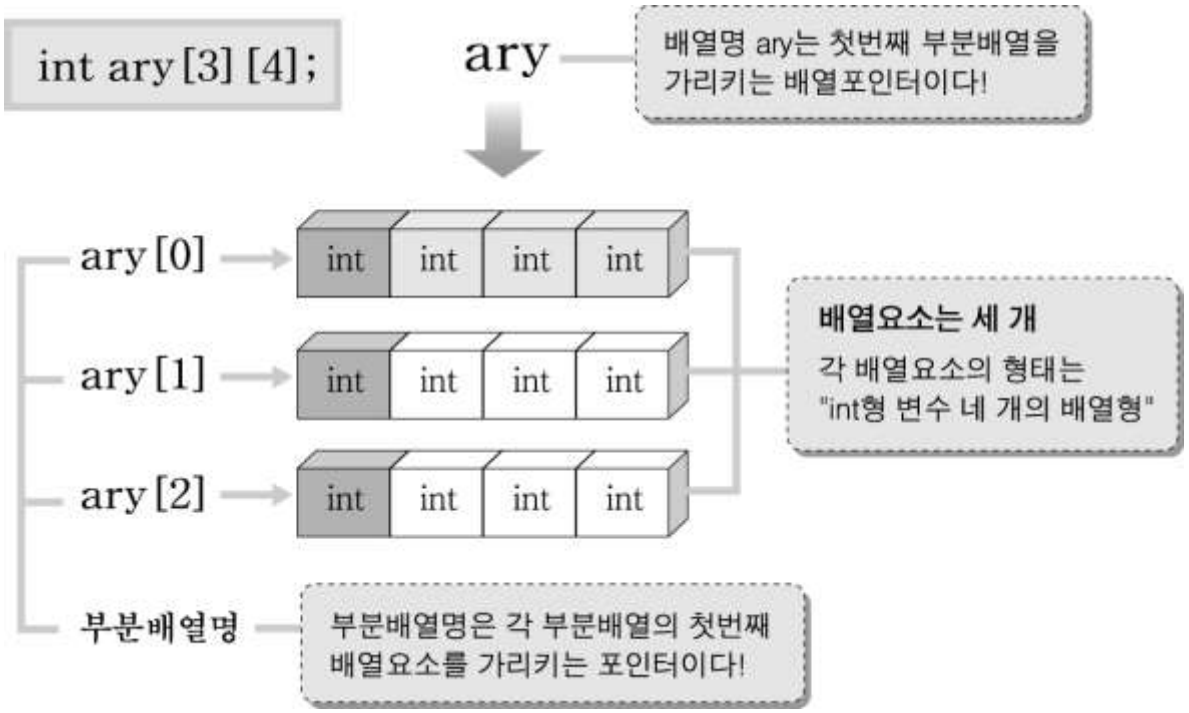
ary + 1 ➡ 1245036 + (1 * sizeof(ary[0])) ➡ 1245036 + (1 * 4) ➡ 1245040

&ary + 1 ➡ 1245036 + (1 * sizeof(ary)) ➡ 1245036 + (1 * 20) ➡ 1245056



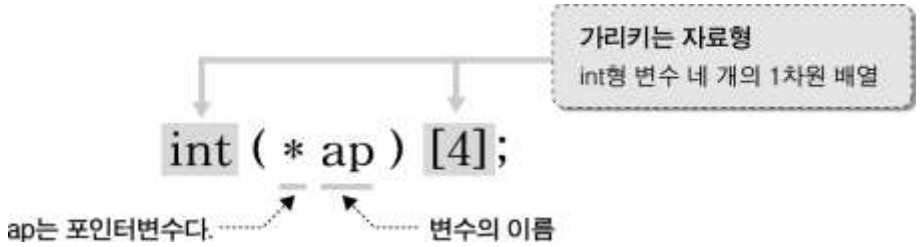
▶ 2차원 배열의 배열명은 배열포인터이다.

- 2차원 배열의 배열명은 첫 번째 부분배열을 가리키는 배열포인터다.



▶ 배열포인터변수

- 배열포인터는 배열포인터변수에 저장한다.



– 배열포인터변수가 2차원 배열의 배열명을 저장하면 2차원 배열처럼 사용된다.

```
int ary[3][4]={{1,2,3,4}, {5,6,7,8}, {9,10,11,12}};
int (*ap)[4];
int i, j;
ap=ary;     // 배열명을 배열포인터변수에 저장
for(i=0; i<3; i++){
    for(j=0; j<4; j++){
        printf("%5d", ap[i][j]);
    }                    // ap를 배열명처럼 사용한다.
    printf("₩n");
}
```

출력 결과

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

▶ 2차원 배열을 처리하는 함수

- 2차원 배열의 배열명을 전달인자로 받는 함수는 매개변수로 배열 포인터변수를 선언한다.

```
void ary_prn( int (*ap)[4] );    // 함수의 선언, 매개변수는 배열포인터변수
```

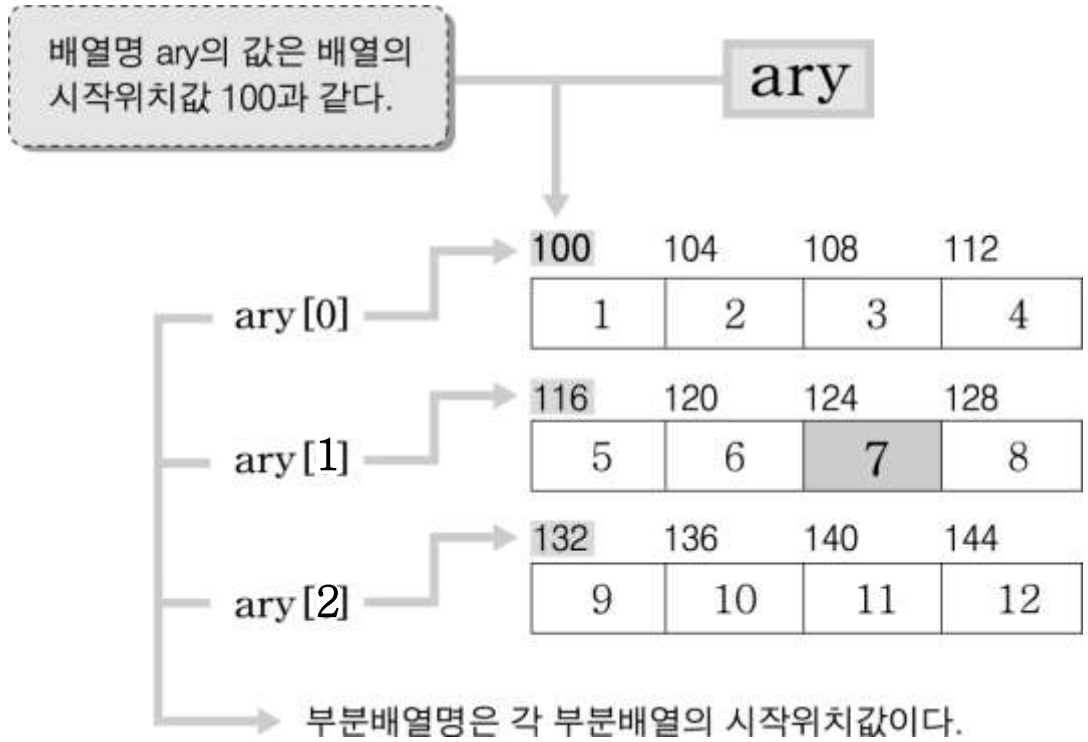
(함수의 선언에는 변수명 생략이 가능하므로 int (*)[4]와 같이 적을 수 있습니다.)

```
#include <stdio.h>
void ary_prn(int (*)[4]);
int main()
{
    int ary[3][4]={{1,2,3,4},
                   {5,6,7,8},
                   {9,10,11,12}};
    ary_prn(ary);
    return 0;
}
```

```
void ary_prn(int (*ap)[4])
{
    int i, j;
    for(i=0; i<3; i++){
        for(j=0; j<4; j++){
            printf("%5d", ap[i][j]);
        }
        printf("₩n");
    }
}
```

▶ 2차원 배열에서 배열명으로 기억공간을 참조하는 과정

- 1차원의 물리적 기억공간을 행렬의 2차원 구조로 참조할 수 있는 것은 2차원 배열의 배열명과 부분배열명이 각각 배열포인터와 포인터로서 적절한 기능을 수행하기 때문이다.



▶ 2차원 배열에서 배열명으로 기억공간을 참조하는 과정

- 7번째 물리적 배열요소를 참조하는 주소계산 과정

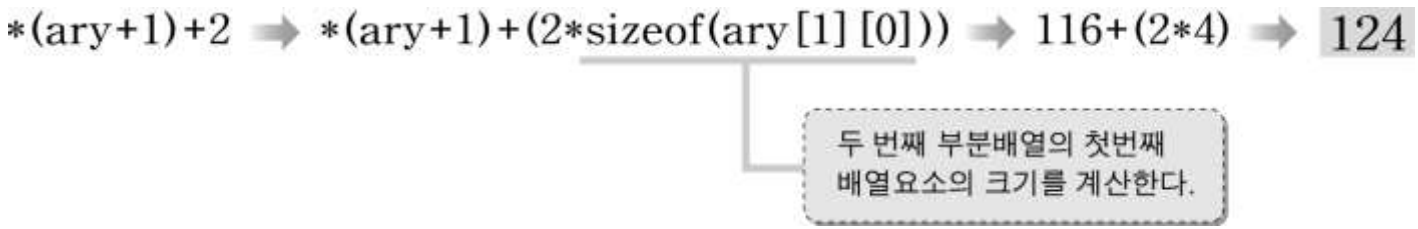
1. 먼저 두 번째 부분배열의 시작위치값을 구한다.

2. 116번지는 두 번째 부분배열을 가리키는 배열포인터이므로 일단 부분배열을 참조하는 과정이 필요하며, 참조의 결과는 부분배열명과 같다.

```
*(ary+1)  ➡  ary[1]    // 두 번째 부분배열명
```

▶ 2차원 배열에서 배열명으로 기억공간을 참조하는 과정

3. 2번 연산의 결과는 5번째 물리적 배열요소를 가리키는 포인터이다.
   여기에 2를 더하여 7번째 물리적 배열요소를 가리키는 포인터를 구한다.

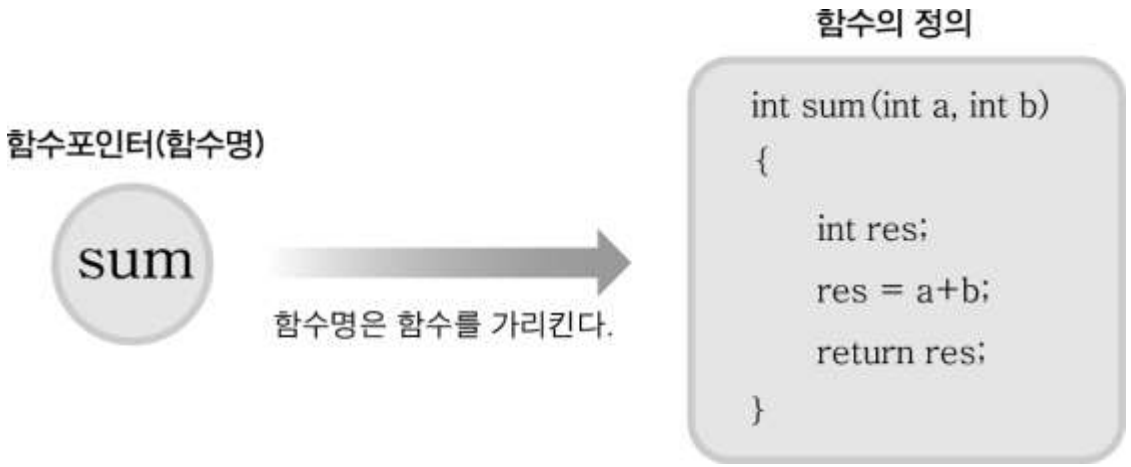


4. 3번 연산의 결과로 구해진 124번지는 7번째 물리적 배열요소를 가리키는
   포인터이므로 참조연산자를 사용하여 그 값을 참조한다.

```
*(*(ary+1)+2)  ➡  ary[1][2]    //두 번째 부분배열의 세 번째 배열요소
```

❑ 함수 포인터

- 함수포인터는 함수의 이름이다.
  - 함수명은 함수의 정의가 있는 메모리의 위치값이며 함수를 가리킨다.



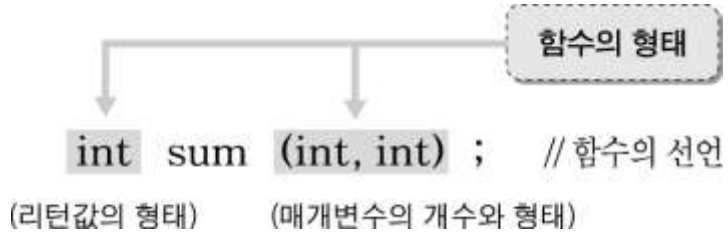
  - 함수명이 포인터라는 증거는 참조연산자를 사용하면 알 수 있다.

```
(*sum)(10, 20);    // 10과 20을 전달인자로 주고 sum함수를 호출한다.
```

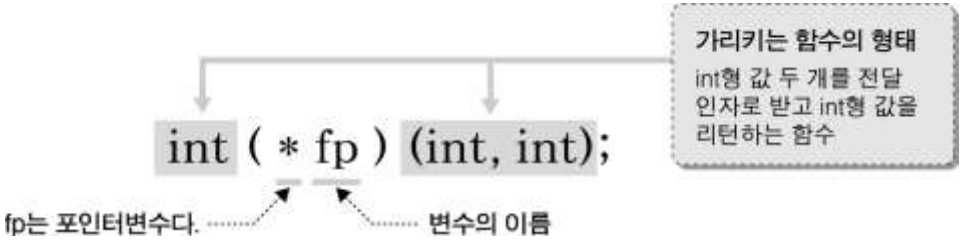
(함수를 참조한 후에 호출해야 하므로 참조연산에 괄호를 사용한다.)

▶ 함수포인터변수

- 함수포인터가 가리키는 자료형은 함수의 형태이다.
  - 함수의 형태를 매개변수의 개수와 형태, 리턴값의 형태이다.



- 함수포인터를 함수포인터변수에 저장하여 호출할 수 있다.




```
int (*fp)(int, int);      // 함수포인터변수 선언
fp=sum;                   // 함수명을 함수포인터변수에 저장한다.
fp(10, 20);               // 함수포인터변수로 함수 호출, (*fp)(10, 20)도 사용 가능
```

▶ 함수포인터변수로 sum함수를 호출하는 프로그램

```
#include <stdio.h>

int sum(int, int);                      // 함수의 선언

int main()
{
    int (*fp)(int, int);                // 함수 포인터변수 선언
    int res;                            // 리턴값을 저장할 변수

    fp=sum;                             // 함수명을 함수포인터변수에 저장한다.
    res=fp(10, 20);                     // 함수포인터변수로 함수를 호출한다.
    printf("result : %d₩n", res);       // 리턴값 출력

    return 0;
}

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

▶ 함수포인터는 어디에 사용하는가?

- 함수포인터변수는 함수의 형태만 같으면 기능과 상관없이 모든 함수 포인터를 저장할 수 있다.

형태가 같다

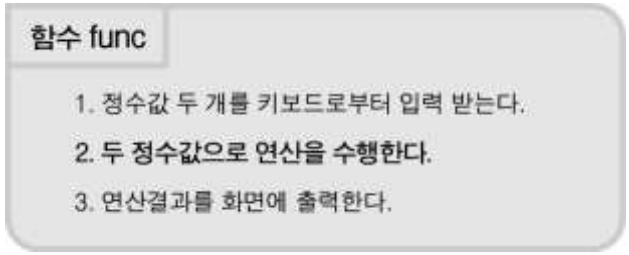
```
int sum(int, int);   // 두 정수값을 더해서 리턴하는 함수
int mul(int, int);   // 두 정수값을 곱해서 리턴하는 함수
int max(int, int);   // 두 정수값 중에서 큰 값을 리턴하는 함수
...
```

```
int (*fp) (int, int);  ➡  fp = sum;
                          fp = mul;   } 모두 사용 가능하다.
                          fp = max;
                          ...
```

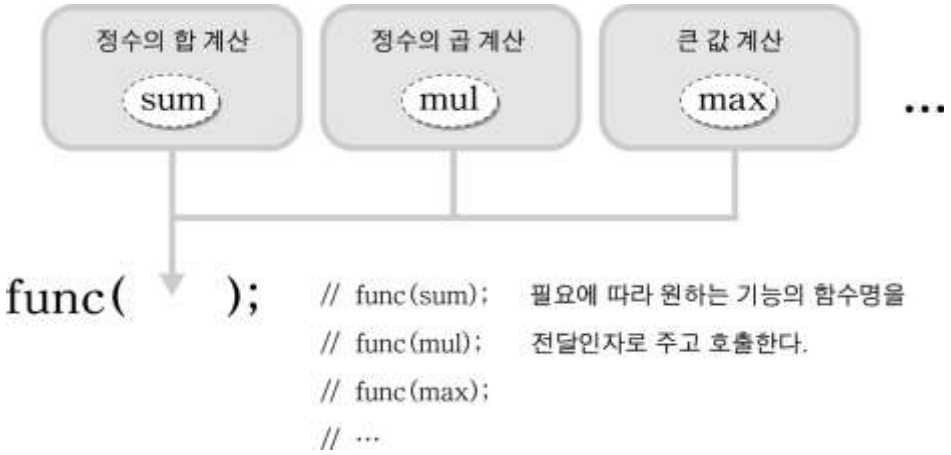
- 형태가 같은 다양한 기능의 함수를 선택적으로 호출하는데 사용할 수 있다.

▶ 함수포인터는 어디에 사용하는가?

- 다음과 같은 기능을 수행하는 함수를 만들자.



– func함수에서 2번 연산과정은 함수를 호출할 때 필요한 연산이 수행되도록 만들자.




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

▶ func함수를 사용한 프로그램

```
#include <stdio.h>
void func(int (*)(int, int));
int sum(int, int);
int mul(int, int);
int max(int, int);
int main()
{
    int sel;
    printf("1. 두 정수의 합₩n");
    printf("2. 두 정수의 곱₩n");
    printf("3. 두 정수 중에서 큰 값 계산₩n");
    printf("원하는 작업을 선택하세요 : ");
    scanf("%d", &sel);
    switch(sel){
    case 1: func(sum); break;
    case 2: func(mul); break;
    case 3: func(max); break;
    }
    return 0;
}
```

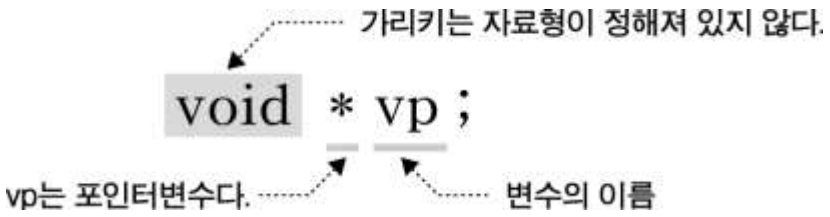
```
void func(int (*fp)(int, int))
{
    int a, b;
    int res;
    printf("두 정수값을 입력하세요 : ");
    scanf("%d%d", &a, &b);
    res=fp(a, b);
    printf("결과값은 : %d₩n", res);
}

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

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

❑ void 포인터

- void포인터는 가리키는 자료형에 대한 정보가 없는 포인터이다.

```
int a;              //int형 변수 선언
(void *) &a;        //int형 변수의 포인터를 void 포인터로 강제 형변환
```

- void포인터변수는 가리키는 자료형이 정해져 있지 않으므로 모든 포인터를 저장할 수 있다.




```
int in;             // int형 변수
double db;          // double형 변수
void *vp;           // void포인터변수

vp=&in;             // int형 변수의 포인터를 저장할 수 있고,
vp=&db;             // double형 변수의 포인터도 저장할 수 있다.
```

▶ void포인터변수의 사용

- 가리키는 자료형에 대한 정보가 없으므로 사용할 때는 형변환하여 사용한다.

```
printf("%d₩n", *(int *)vp);     // 형변환 후에 참조연산자로 저장된 값을 출력한다.
vp=(int *)vp+1;                 // 형변환 후에 정수값을 더한다.
int *ip=(int *)vp;              // 형변환 후에 포인터변수에 대입한다.
```

- void포인터변수를 사용하면 다양한 형태의 포인터를 전달인자로 받을 수 있는 함수를 만들 수 있다.

```
void exchange(void *, void *);   // 함수의 선언

int a=10, b=20;
double x=0.15, y=0.5;
exchange(&a, &b);     // int형 포인터를 전달인자로 주고 호출한다.
exchange(&x, &y);     // double형 포인터를 전달인자로 준다.
```

▶ 정수와 실수를 모두 교환할 수 있는 함수 프로그램

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

void exchange(char *, void *, void *);

int main()
{
    int a=10, b=20;
    double da=1.5, db=2.5;

    exchange("int", &a, &b);
    printf("정수값 교환후 : %d, %d₩n", a, b);
    exchange("double", &da, &db);
    printf("실수값 교환후 : %.1lf, %.1lf₩n", da, db);

    return 0;
}
```

```
void exchange(char *type, void *vp1, void *vp2)
{                          // type은 자료형 구분을 위한 매개변수
    int itp;
    double dtp;

    if(strcmp(type, "int")==0){
        itp=*(int *)vp1;
        *(int *)vp1 = *(int *)vp2;
        *(int *)vp2 = itp;
    }
    if(strcmp(type, "double")==0){
        dtp=*(double *)vp1;
        *(double *)vp1 = *(double *)vp2;
        *(double *)vp2 = dtp;
    }
}
```

type에 따라 형변환하여 사용