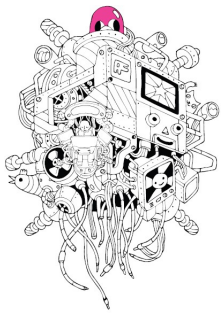
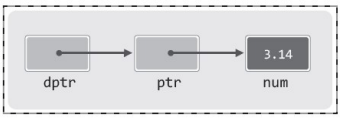


Chapter 17. 포인터의 포인터

Chapter 17-1. 포인터의 포인터에 대한 이해

포인터 변수를 가리키는 이중 포인터 변수

```
int main(void)
{
    double num=3.14;
    double * ptr=&num;
    double ** dptr =&ptr;
    . . . . .
}
```



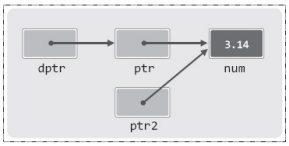
포인터 변수의 주소 값을 저장하는 것이 **이중 포인터 변수(더블 포인터 변수)**이다.

위의 상황에서 *dptr은 포인터 변수 ptr을...

*(*dptr)은 변수 num을 의미하게 된다.

DoublePointerAccess.c

```
int main(void)
{
    double num = 3.14;
    double *ptr = &num;
    double **dptr = &ptr;
    double *ptr2;

    printf("%9p %9p \n", ptr, *dptr);
    printf("%9g %9g \n", num, **dptr);
    ptr2 = *dptr;  // ptr2 = ptr 과 같은 문장
    *ptr2 = 10.99;
    printf("%9g %9g \n", num, **dptr);
    return 0;
}
```



이 상황에서 변수 num에 접근하는 네 가지 방법은?

```
0032FD00    0032FD00
    3.14        3.14
   10.99       10.99
```

실행결과

포인터 변수의 Swap 1

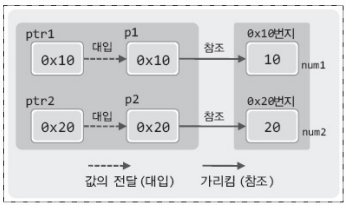
PointerSwapFail.c

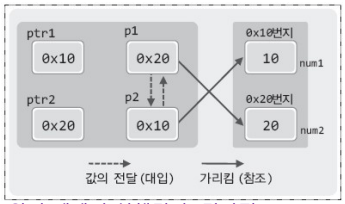
```
void SwapIntPtr(int *p1, int *p2)
{
    int * temp=p1;
    p1=p2;
    p2=temp;
}

int main(void)
{
    int num1=10, num2=20;
    int *ptr1, *ptr2;
    ptr1=&num1, ptr2=&num2;
    printf("*ptr1, *ptr2: %d %d \n", *ptr1, *ptr2);

    SwapIntPtr(ptr1, ptr2);
    printf("*ptr1, *ptr2: %d %d \n", *ptr1, *ptr2);
    return 0;
}
```

ptr1과 ptr2의 swap은 성공하는가? 문제점은?

```
*ptr1, *ptr2: 10 20
*ptr1, *ptr2: 10 20
```

실행결과





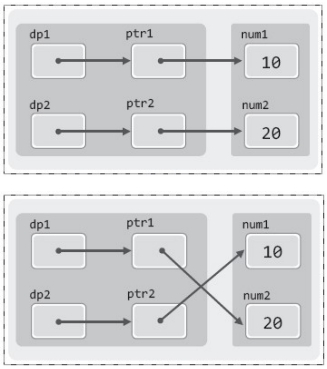
왼편 예제의 실행결과! 결과적으로 ptr1과 ptr2에 저장된 값은 서로 바뀌지 않는다.

포인터 변수의 Swap 2

PointerSwapSuccess.c

```
void SwapIntPtr(int **dp1, int **dp2)
{
    int *temp = *dp1;
    *dp1 = *dp2;
    *dp2 = temp;
}

int main(void)
{
    int num1=10, num2=20;
    int *ptr1, *ptr2;
    ptr1=&num1, ptr2=&num2;
    printf("*ptr1, *ptr2: %d %d \n", *ptr1, *ptr2);

    SwapIntPtr(&ptr1, &ptr2);  // ptr1과 ptr2의 주소 값 전달!
    printf("*ptr1, *ptr2: %d %d \n", *ptr1, *ptr2);
    return 0;
}
```

포인터 변수에 저장된 값의 변경이 목적이므로 포인터 변수의 주소 값을 함수에 전달해야 한다.



이중 포인터를 이용해서 두 포인터 변수의 swap에 성공한다.

```
*ptr1, *ptr2: 10 20
*ptr1, *ptr2: 20 10
```

실행결과

포인터 배열과 포인터 배열의 포인터 형

```
int * arr1[20];
double * arr2[30];
```

Ch 13의 후반에 학습한 포인터 변수로 이뤄진 배열(포인터 배열)

int arr1[3]; 에서 arr1의 포인터 형은 int * double arr2[3]; 에서 arr2의 포인터 형은 double *

이렇듯 1차원 배열이름의 포인터 형은 배열 이름이 가리키는 대상을 기준으로 결정된다.

따라서 int *arr1[20]; 에서 arr1의 포인터 형은 int **

double *arr2[30]; 에서 arr2의 포인터 형은 double **

PointerArrayType.c

```
int main(void)
{
    int num1=10, num2=20, num3=30;
    int *ptr1=&num1;
    int *ptr2=&num2;
    int *ptr3=&num3;

    int * ptrArr[]={ptr1, ptr2, ptr3};
    int **dptr=ptrArr;

    printf("%d %d %d \n", *(ptrArr[0]), *(ptrArr[1]), *(ptrArr[2]));
    printf("%d %d %d \n", *(dptr[0]), *(dptr[1]), *(dptr[2]));
    return 0;
}
```

```
10 20 30
10 20 30
```

실행결과

Chapter 17-2. 다중 포인터 변수와 포인터의 필요성

이중 포인터를 가리키는 삼중 포인터

```
int ***tptr;
```

삼중 포인터 변수!
이중 포인터 변수의 주소 값을 담는 용도로 선언된다.

TriplePointer.c

```
int main(void)
{
    int num=100;
    int *ptr=&num;
    int **dptr=&ptr;
    int ***tptr=&dptr;

    printf("%d %d \n", **dptr, ***tptr);
    return 0;
}
```

실행결과

```
100 100
```

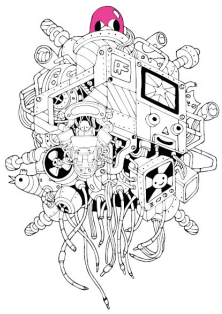
이중 포인터 변수의 개념을 그대로 확장해서 이해할 수 있는 것이 **삼중 포인터 변수**이다!

포인터의 필요성은 어디서 찾아야 하는가?

- scanf 함수와 같이 함수 내에서 **함수 외부에 선언된 변수의 접근**을 허용하기 위함
- **메모리의 동적 할당** 등 PART 04에서 공부하는 내용을 통해서 포인터의 필요성을 다양하게 이해하게 된다.
- 향후에 **자료구조**라는 과목을 공부하게 되면 보다 넓게 필요성을 이해할 수 있게 된다.

실습 문제 (Lab 1)

- **이중 포인터 변수의 활용**
  - 다음과 같은 두 개의 int형 포인터 변수와 길이가 5인 int형 배열을 선언한다.
  - int *maxPtr;
  - int *minPtr;
  - int arr[5];
  - 그리고 "MaxAndMin"이란 이름의 함수를 정의하고 이를 호출하면서 위의 배열과 두 포인터 변수에 대한 정보를 전달
    - 함수 호출 형태: **MaxAndMin (arr, sizeof(arr)/sizeof(int), &maxPtr, &minPtr);**
  - 함수 호출이 완료되면 포인터 변수 maxPtr에는 가장 큰 값이 저장된 배열요소의 주소 값이, minPtr에는 가장 작은 값이 저장된 배열요소의 주소 값이 저장되어야 함



Chapter 18. 다차원 배열과 포인터의 관계

Chapter 18-1. 2차원 배열 이름의 포인터 형

1차원 배열이름과 2차원 배열이름의 포인터 형

```
int arr[10];
```

Int 형 1차원 배열이므로 arr은 int형 포인터 (int *)

```
int * parr[20];
```

Int 포인터 형 1차원 배열이므로 parr은 int형 이중 포인터 (int **)

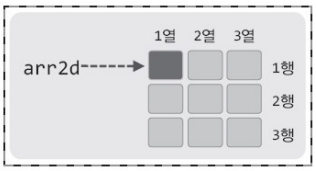
```
int arr2d[3][4];
```

int형 1차원 배열도, int 포인터 형 1차원 배열도 아니므로 arr2d는 int형 포인터 형도,
int형 이중 포인터 형도 아니다. 2차원 배열이름의 포인터 형을 결정짓는 방법은 별도로 존재한다.

2차원 배열이름이 가리키는 것들은?

2차원 배열이름의 포인터 형을 결정지으려면 우선 2차원 배열이름이 가리키는 대상이 무엇인지 알아야 한다. 그런데 1차원 배열과 달리 이것만으로 포인터 형이 결정되지 않는다.

int arr2d[3][3];



배열이름 arr2d가 가리키는 것은 인덱스 기준으로 [0][0]에 위치한 첫 번째 요소

2차원 배열의 경우 arr2d[0], arr2d[1], arr2d[2]도 의미를 지닌다. 각각 1행, 2행, 3행의 첫 번째 요소를 가리키는 주소값의 의미를 지닌다.

그럼 arr2d와 arr2d[0]는 같은 것인가?

2DArrayAddress.c

```
int main(void)
{
    int arr2d[3][3];
    printf("%d \n", arr2d);
    printf("%d \n", arr2d[0]);
    printf("%d \n\n", &arr2d[0][0]);

    printf("%d \n", arr2d[1]);
    printf("%d \n\n", &arr2d[1][0]);

    printf("%d \n", arr2d[2]);
    printf("%d \n\n", &arr2d[2][0]);

    printf("sizeof(arr2d): %d \n", sizeof(arr2d));
    printf("sizeof(arr2d[0]): %d \n", sizeof(arr2d[0]));
    printf("sizeof(arr2d[1]): %d \n", sizeof(arr2d[1]));
    printf("sizeof(arr2d[2]): %d \n", sizeof(arr2d[2]));
    return 0;
}
```

실행결과

```
4585464
4585464
4585464

4585476
4585476

4585488
4585488

sizeof(arr2d): 36
sizeof(arr2d[0]): 12
sizeof(arr2d[1]): 12
sizeof(arr2d[2]): 12
```

배열이름 arr2d를 대상으로 sizeof 연산을 하는 경우 배열 전체의 크기를 반환

arr2d[0], arr2d[1], arr2d[2]를 대상으로 sizeof 연산을 하는 경우 각 행의 크기를 반환

배열이름 기반의 포인터 연산

```
int iarr[3];        // iarr은 int형 포인터
double darr[7];     // darr은 double형 포인터
```

```
printf("%p", iarr+1);
printf("%p", darr+1);
```

iarr은 int형 포인터이기 때문에 +1의 결과로 sizeof(int)의 크기만큼 값이 증가한다.

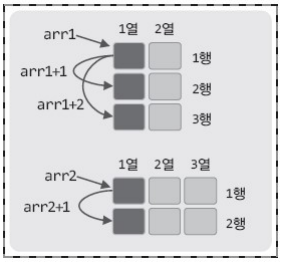
darr은 double형 포인터이기 때문에 +1의 결과로 sizeof(double)의 크기만큼 값이 증가한다.

이렇듯 포인터 연산의 결과는 포인터 형에 의존적이다. 따라서 2차원 배열이름의 포인터 형을 결정짓기 위한 힌트는 포인터 연산의 결과를 주목하면 얻을 수 있다.

2차원 배열이름 대상의 포인터 연산 결과

2DArrPointerOp.c

```
int main(void)
{
    int arr1[3][2];
    int arr2[2][3];

    printf("arr1: %p \n", arr1);
    printf("arr1+1: %p \n", arr1+1);
    printf("arr1+2: %p \n\n", arr1+2);

    printf("arr2: %p \n", arr2);
    printf("arr2+1: %p \n", arr2+1);
    return 0;
}
```



2차원 배열이름을 대상으로 값을 1씩 증가 및 감소하는 경우 그 결과는 **각 행의 첫 번째 요소의 주소** 값이다.

```
arr1: 004BFBE0
arr1+1: 004BFBE8
arr1+2: 004BFBF0

arr2: 004BFBC0
arr2+1: 004BFBCC
```

실행결과

arr1과 arr2는 둘 다 int형 2차원 배열이다. 그러나 가로의 길이가 다르기 때문에 포인터 연산결과로 증가 및 감소하는 값의 크기에는 차이가 있다. 즉, arr1과 arr2의 포인터 형은 일치하지 않는다.

이렇듯 2차원 배열이름의 포인터 형은 배열의 가로길이에 따라서도 나뉜다. 그리고 이러한 특징 때문에 2차원 배열이름의 포인터 형 결정이 쉽지 않은 것이다.

최종결론! 2차원 배열이름의 포인터 형(1/2)

1. 2차원 배열이름의 포인터 형을 결정짓는 두 가지 요소!

1. **가리키는 대상**은 무엇인가?
2. 배열이름(포인터)를 대상으로 값을 1 증가 및 감소 시 **얼마 만큼 증가 및 감소**하는가?

2. 왜 다른가?

1차원 배열이름의 자료형은 1차원 배열이름이 가리키는 대상만으로 결정이 되는데 2차원 배열이름의 자료형은 그렇지 않은 이유는?

3. 포인터 형을 통한 메모리의 접근과 주소 값의 증가

1차원 배열의 경우에는 배열이름이 가리키는 대상을 기준으로 메모리의 접근방법과 포인터 연산시의 증가 및 감소의 크기가 결정되었다. 그러나 2차원 배열에서는 위의 두 가지 정보가 모두 존재해야 이 둘을 결정지을 수 있다.

최종결론! 2차원 배열이름의 포인터 형(2/2)

int arr[3][4]의 포인터 형은?

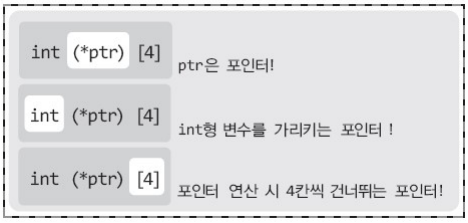
어색하지만 이것이 arr의 포인터 형을 설명하는 최선의 방법이다. int형 포인터, double형 포인터와 같이 딱 떨어지는 명칭이 존재하지 않는다.

1. 가리키는 대상 **int형 변수**
2. 포인터 연산의 결과 **sizeof(int)×4**의 크기단위로 값이 증가 및 감소

이러한 유형의 포인터 변수 ptr의 선언

int (*ptr) [4];

2차원 배열을 가리키는 포인터 변수이므로 **배열 포인터** 변수라 한다.



2차원 배열이름의 포인터 형 결정짓는 연습

case 1

```
char (*arr1)[4];
double (*arr2)[7];
```

"arr1은 char형 변수를 가리키면서, 포인터 연산 시 sizeof(char)×4의 크기단위로 값이 증가 및 감소하는 포인터 변수"

"arr2는 double형 변수를 가리키면서, 포인터 연산 시 sizeof(double)×7의 크기단위로 값이 증가 및 감소하는 포인터 변수"

case 2

"int형 변수를 가리키면서, 포인터 연산 시 sizeof(int)×2의 크기단위로 값이 증가 및 감소하는 포인터 변수 ptr1"

"float형 변수를 가리키면서, 포인터 연산 시 sizeof(float)×5의 크기단위로 값이 증가 및 감소하는 포인터 변수 ptr2"

```
int (*ptr1)[2];
float (*ptr2)[5];
```

2차원 배열이름의 포인터 관련 예제

2DArrNameAndArrPtr.c

```
int main(void)
{
    int arr1[2][2]={
        {1, 2}, {3, 4}
    };
    int arr2[3][2]={
        {1, 2}, {3, 4}, {5, 6}
    };
    int arr3[4][2]={
        {1, 2}, {3, 4}, {5, 6}, {7, 8}
    };

    int (*ptr)[2];
    int i;

    ptr=arr1;
    printf("** Show 2,2 arr1 **\n");
    for(i=0; i<2; i++)
            printf("%d %d \n", ptr[i][0], ptr[i][1]);

    ptr=arr2;
    printf("** Show 3,2 arr2 **\n");
    for(i=0; i<3; i++)
            printf("%d %d \n", ptr[i][0], ptr[i][1]);

    ptr=arr3;
    printf("** Show 4,2 arr3 **\n");
    for(i=0; i<4; i++)
            printf("%d %d \n", ptr[i][0], ptr[i][1]);
    return 0;
}
```

arr1, arr2, arr3는 둘 다 int형 2차원 배열이면서 가로 길이가 같으므로 포인터 형이 동일하다.

실행결과

```
** Show 2,2 arr1 **
1 2
3 4
** Show 3,2 arr2 **
1 2
3 4
5 6
** Show 4,2 arr3 **
1 2
3 4
5 6
7 8
```

실습 문제 (Lab 1)

- **이중 포인터 변수의 활용**
  - 다음과 같은 두 개의 int형 포인터 변수와 길이가 5인 int형 배열을 선언한다.
  - int *maxPtr;
  - int *minPtr;
  - int arr[5];
  - 그리고 "MaxAndMin"이란 이름의 함수를 정의하고 이를 호출하면서 위의 배열과 두 포인터 변수에 대한 정보를 전달
    - 함수 호출 형태: **MaxAndMin (arr, sizeof(arr)/sizeof(int), &maxPtr, &minPtr);**
  - 함수 호출이 완료되면 포인터 변수 maxPtr에는 가장 큰 값이 저장된 배열요소의 주소 값이, minPtr에는 가장 작은 값이 저장된 배열요소의 주소 값이 저장되어야 함

실습 문제 (숙박관리 프로그램 확장 1)

- 숙박관리 프로그램 작성 (Version 2)
  - 목적: Chapter 16의 Lab 2 프로그램 확장 (각 메뉴 처리를 함수로 구현)
  - 입실처리 함수: Check_in (room, sizeof(room)/sizeof(room[0]);
  - 퇴실처리 함수: Check_out (room, sizeof(room)/sizeof(room[0]);
  - 조회처리 함수: List_print (room, sizeof(room)/sizeof(room[0]);

```
**********************
** 숙박관리 프로그램 **
**********************
1.입실, 2.퇴실, 3.조회, 4.종료:  1
층 번호 입력(1~3층): 1
방 번호 입력(1~5호): 1
고객이름(1자): A


1.입실, 2.퇴실, 3.조회, 4.종료:  3
A       0       0       0       0
0       0       0       0       0
0       0       0       0       0


1.입실, 2.퇴실, 3.조회, 4.종료:
```

실습 시간 (2019년 10월 8일)

- 예제1 (17장): DoublePointerAccess.c, PointerArrayType.c, (PointSwapFail.c), PointSwapSuccess.c, (TriplePointer.c)
- 예제2 (18장): 2DArrayAddress.c, 2DArrPointerOp.c, 2DArrNameAndArrPtr.c

- Lab 문제: Lab1.c, 숙박관리 프로그램 (ver.2)

Chapter 18-2. 2차원 배열이름의 특성과 주의사항

‘배열 포인터’와 ‘포인터 배열’을 혼동하지 말자

```
int * whoA [4];     // 포인터 배열
int (*whoB) [4];    // 배열 포인터
```

포인터 배열 포인터 변수로 이루어진 배열

배열 포인터 배열을 가리킬 수 있는 포인터 변수

ArrPtrAndPtrArr.c

```
int main(void)
{
    int num1=10, num2=20, num3=30, num4=40;
    int arr2d[2][4]={1, 2, 3, 4, 5, 6, 7, 8};
    int i, j;

    int * whoA[4]={&num1, &num2, &num3, &num4};   // 포인터 배열
    int (*whoB)[4]=arr2d;   // 배열 포인터

    printf("%d %d %d %d \n", *whoA[0], *whoA[1], *whoA[2], *whoA[3]);
    for(i=0; i<2; i++)
    {
        for(j=0; j<4; j++)
            printf("%d ", whoB[i][j]);
        printf("\n");
    }
    return 0;
}
```

실행결과

```
10 20 30 40
1 2 3 4
5 6 7 8
```

2차원 배열을 함수의 인자로 전달하기

```
int main(void)
{
    int arr1[2][7];      <-------> int (*parr1)[7]
    double arr2[4][5];   <-------> double (*parr2)[5]
    SimpleFunc(arr1, arr2);
    . . . .
}
```

매개변수의 선언 위치에서만 동일한 선언으로 간주된다.

```
void SimpleFunc( int (*parr1)[7], double (*parr2)[5] ) { . . . . }
                  ↕ 동일한 선언          ↕ 동일한 선언
void SimpleFunc( int parr1[][7], double parr2[][5] ) { . . . . }
```

2차원 배열을 함수의 인자로 전달하는 예제

```
void ShowArr2DStyle(int (*arr)[4], int column)
{                              // 배열요소 전체출력
    int i, j;
    for(i=0; i<column; i++)
    {
        for(j=0; j<4; j++)
            printf("%d ", arr[i][j]);
        printf("\n");
    }
    printf("\n");
}

int Sum2DArr(int arr[][4], int column)
{                          // 배열요소의 합 반환
    int i, j, sum=0;
    for(i=0; i<column; i++)
        for(j=0; j<4; j++)
            sum += arr[i][j];
    return sum;
}
```

```
1 2 3 4
5 6 7 8

1 1 1 1
3 3 3 3
5 5 5 5

arr1의 합: 36
arr2의 합: 36
```

실행결과

정의된 두 함수의 인자로 전달되는 2차원 배열의 가로길이는 결정되어 있다.

반면, 세로 길이 정보는 결정되어 있지 않고 두 번째 인자를 통해서 추가로 전달하고 있다. 이점에 주목하자!

2DArrParam.c

```
int main(void)
{
    int arr1[2][4]={1, 2, 3, 4, 5, 6, 7, 8};
    int arr2[3][4]={1, 1, 1, 1, 3, 3, 3, 3, 5, 5, 5, 5};
                                                  배열의 세로길이 계산방식
    ShowArr2DStyle(arr1, sizeof(arr1)/sizeof(arr1[0]));
    ShowArr2DStyle(arr2, sizeof(arr2)/sizeof(arr2[0]));
    printf("arr1의 합: %d \n", Sum2DArr(arr1, sizeof(arr1)/sizeof(arr1[0])));
    printf("arr2의 합: %d \n", Sum2DArr(arr2, sizeof(arr2)/sizeof(arr2[0])));
    return 0;
}
```

2차원 배열에서도 arr[i]와 *(arr+i)는 같다.

```
arr[i] == *(arr+i)
```

Ch13에서 1차원 배열과 포인터 변수를 대상으로 내린 결론! 2차원 배열에서도 그대로 적용이 된다!

```
int arr[3][2]={ {1, 2}, {3, 4}, {5, 6} };
```

```
arr[2][1]=4;
(*(arr+2))[1]=4;
*(arr[2]+1)=4;
*(*(arr+2)+1)=4;
```

위에 선언된 배열 arr을 대상으로 인덱스 기준 [2][1] 번째 요소에 4를 저장하는, 모두 동일한 결과를 보이는 문장이다.

2DArrAccessType.c

```
int main(void)
{
    int a[3][2]={{1, 2}, {3, 4}, {5, 6}};
    printf("a[0]: %p \n", a[0]);
    printf("*(a+0): %p \n", *(a+0));
    printf("a[1]: %p \n", a[1]);
    printf("*(a+1): %p \n", *(a+1));
    printf("a[2]: %p \n", a[2]);
    printf("*(a+2): %p \n", *(a+2));
    printf("%d, %d \n", a[2][1], (*(a+2))[1]);
    printf("%d, %d \n", a[2][1], *(a[2]+1));
    printf("%d, %d \n", a[2][1], *(*(a+2)+1));
    return 0;
}
```

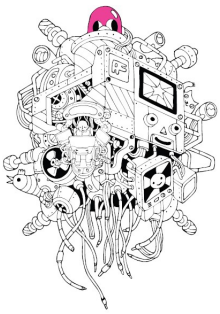
실행결과

```
a[0]: 001AFDC8
*(a+0): 001AFDC8
a[1]: 001AFDD0
*(a+1): 001AFDD0
a[2]: 001AFDD8
*(a+2): 001AFDD8
6, 6
6, 6
6, 6
```

2차원 배열 인덱스 사용 연습

- 다음 예제의 출력 결과는?

```
int main(void)
{
    int arr[3][2] = {{1, 2}, {3, 4}, {5, 6}};
    printf("%d %d ₩n", arr[1][0], arr[0][1]);
    printf("%d %d ₩n", *(arr[2]+1), *(arr[1]+1));
    printf("%d %d ₩n", (*(arr+2))[0], (*(arr+0))[1]);
    printf("%d %d ₩n", **arr, *(*(arr+0)+0));

    return 0;
}
```



Chapter 19. 함수 포인터와 void 포인터

Chapter 19-1. 함수 포인터와 void 포인터

함수 포인터의 이해

1. 함수 포인터

1. 함수의 이름은 함수가 저장된 메모리 공간을 가리키는 포인터이다(함수 포인터).
2. 함수의 이름이 의미하는 주소 값은 함수 포인터 변수를 선언해서 저장할 수 있다.
3. 함수 포인터 변수를 선언하라면 함수 포인터의 형(type)을 알아야 한다.

2. 함수 포인터의 형(type)

1. 함수 포인터의 형 정보에는 반환형과 매개변수 선언에 대한 정보를 담기로 약속
2. 즉, 함수의 반환형과 매개변수 선언이 동일한 두 함수의 함수 포인터 형은 일치한다.

3. 함수 포인터 형 결정

int SimpleFunc(int num) **반환형 int, 매개변수 int형 1개**

double ComplexFunc(double num1, double num2) **반환형 double, 매개변수 double형 2개**

적절한 함수 포인터 변수의 선언

```
int (*fptr) (int)          fptr은 포인터!
int (*fptr) (int)          반환형이 int인 함수 포인터!
int (*fptr) (int)          매개변수 선언이 int 하나인 함수 포인터!
```

함수 포인터 변수를 선언하는 방법

```
int SoSimple(int num1, int num2) { . . . . }
int (*fptr) (int, int);      SoSimple 함수이름과 동일한 형의 변수 선언
fptr=SoSimple;               상수의 값을 변수에 저장
fptr(3, 4);    // SoSimple(3, 4)와 동일한 결과를 보임
                             함수 포인터 변수에 저장된 값을 통해서도 함수호출 가능!
```

함수 포인터 변수 관련 예제

```
void SimpleAdder(int n1, int n2)
{
    printf("%d + %d = %d \n", n1, n2, n1+n2);
}
void ShowString(char * str)
{
    printf("%s \n", str);
}
int main(void)
{
    char * str="Function Pointer";
    int num1=10, num2=20;

    void (*fptr1)(int, int) = SimpleAdder;
    void (*fptr2)(char *) = ShowString;

    /* 함수 포인터 변수에 의한 호출 */
    fptr1(num1, num2);
    fptr2(str);
    return 0;
}
```

(1) FuntionPointer.c

(2) UsefulFunctionPointer.c

```
10 + 20 = 30
Function Pointer
```

실행결과

형(Type)이 존재하지 않는 void 포인터

```
void * ptr;
```

어떠한 주소 값도 저장이 가능한 void형 포인터

형 정보가 존재하지 않는 포인터 변수이기에 어떠한 주소 값도 저장이 가능하다.

형 정보가 존재하지 않기 때문에 메모리 접근을 위한 * 연산은 불가능하다.

VoidTypePointer.c

```
void SoSimpleFunc(void)
{
    printf("I'm so simple");
}

int main(void)
{
    int num=20;
    void * ptr;

    ptr=&num;    // 변수 num의 주소 값 저장
    printf("%p \n", ptr);

    ptr=SoSimpleFunc;    // 함수 SoSimpleFunc의 주소 값 저장
    printf("%p \n", ptr);
    return 0;
}
```

```
int main(void)
{
    int num=20;
    void * ptr=&num;
    *ptr=20;    // 컴파일 에러!
    ptr++;    // 컴파일 에러!
    . . . .
}
```

형 정보가 존재하지 않으므로!!

```
001AF974
00F61109
```

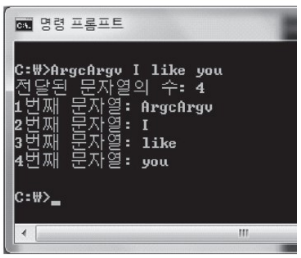
실행결과

Chapter 19-2. main 함수로의 인자 전달

main 함수를 통한 인자의 전달

ArgcArgv.c

```
int main(int argc, char *argv[])
{
    int i=0;
    printf("전달된 문자열의 수: %d \n", argc);

    for(i=0; i<argc; i++)
        printf("%d번째 문자열: %s \n", i+1, argv[i]);
    return 0;
}
```



인자를 전달하는 방식

char * argv[]

```
void SimpleFunc(TYPE * arr) { . . . . }
void SimpleFunc(TYPE arr[]) { . . . . }
```

매개 변수 선언에서는 예외적으로 *arr을 arr[]으로 대신할 수 있다!
앞서 두 차례 확인한 내용!

그대로 적용한다.

```
void SimpleFunc(char **arr) { . . . . }
void SimpleFunc(char * arr[]) { . . . . }
```

즉, char * arr[]는 char형 이중 포인터이다.

char * argv[] 관련 예제

```
void ShowAllString(int argc, char * argv[])
{
    int i;
    for(i=0; i<argc; i++)
        printf("%s \n", argv[i]);
}

int main(void)
{
    char * str[3]={
        "C Programming",
        "C++ Programming",
        "JAVA Programming"
    };
    ShowAllString(3, str);
    return 0;
}
```

ArgvParamType.c

문자열의 주소 값을 모은 배열이므로 **char형 포인터 배열**을 선언!
str의 포인터 형은 char**

```
C Programming
C++ Programming
JAVA Programming
```

실행결과

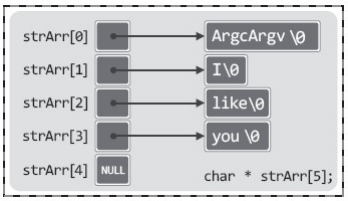
인자의 형성과정

```
c:\>ArgcArgv I like you
```

문자열의 구분

문자열 1	"ArgcArgv"
문자열 2	"I"
문자열 3	"like"
문자열 4	"you"

문자열의 구성



ArgvEndNull.c

```
int main(int argc, char *argv[])
{
    int i=0;
    printf("전달된 문자열의 수: %d \n", argc);

    while(argv[i]!=NULL)
    {
        printf("%d번째 문자열: %s \n", i+1, argv[i]);
        i++;
    }
    return 0;
}
```

실행결과

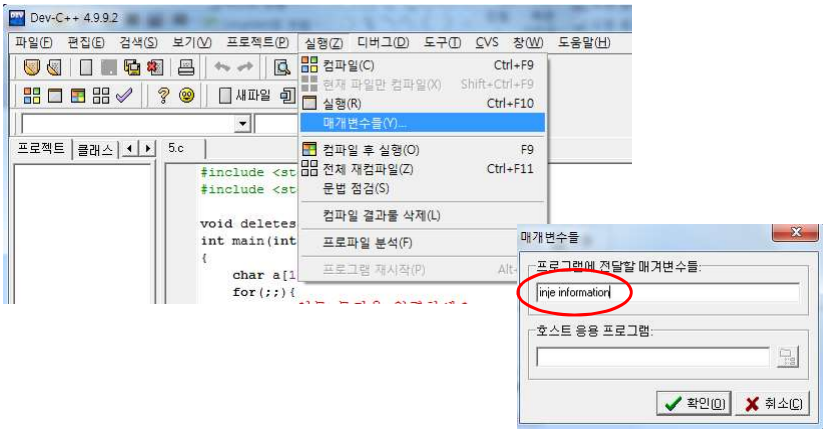
```
C:\> ArgvEndNULL "I love you"
전달된 문자열의 수: 2
1번째 문자열: ArgvEndNULL
2번째 문자열: I love you
```

문자열 기반 함수의 호출

```
main(4, strArr);
```

DEV-C++에서의 명령어 인자 설정하는 법(불필요)

- 명령어 인자를 설정



간단한 DOS 명령어 소개

- "윈도우 검색>cmd 실행 " 또는 "Windows 시스템>명령 프롬프트" 실행

- 간단한 DOS 명령어
  - **dir** : 현재 폴더 또는 디렉토리 보여주기
    - Ex: C:₩users₩admin>dir 또는 dir /w
  - **cd** (의미: change director) : 폴더 또는 디렉토리 변경하기
    - dir로 검색했더니 [Music]이란 폴더가 있을 경우: cd Music [enter]
  - ".": 현재 폴더 또는 디렉토리를 의미 → cd . : 제자리
  - "..": 상위 디렉토리를 의미
    - C:₩users₩admin₩Music>cd .. [enter]
    - C:₩users₩admin>
  - **help**: 도움말

실습문제 (연습 1)

- **명령어 라인 인자로 inje와 information을 받아 들여 인자의 개수와 인가의 값을 출력하는 프로그램**
  - **아래와 같이 출력**

입력된 인자의 개수는 2개입니다.
입력된 인자의 값은 inje와 information입니다.

44

실습문제 (연습 1: Solution)

- 명령어 라인 인자로 inje와 information을 받아 들여 인자의 개수와 인자의 값을 출력하는 프로그램
  - 아래와 같이 출력

```
입력된 인자의 개수는 2개입니다.
입력된 인자의 값은 inje와 information입니다.
```

  - 프로그램

```
void main (int argc, char* argv[ ]) {
    int i;
    printf("입력된 인자의 개수는 %d입니다.\n", argc-1);
    printf("입력된 인자의 값은 %s와 %s입니다.\n", argv[1], argv[2]);
}
```

➔ 개선할 점은??

45

실습문제 (연습 2)

- 명령어 라인 인자로 숫자를 받아 들여 그 만큼의 숫자를 더하는 프로그램
  - 명령어 라인 인자로 1부터 9 사이의 정수 하나 (예, a)를 입력
  - 이후 키보드로부터 a개의 정수를 입력
  - 입력 받은 정수를 모두 더하여 그 합을 출력

```
C:> test.exe 2
정수 값 2개를 입력하세요: 10 20
전체 합계는 30입니다.
```

46

실습문제 (연습 2: Solution)

- 명령어 라인 인자로 숫자를 받아 들여 그 만큼의 숫자를 더하는 프로그램
  - 명령어 라인 인자로 1부터 9 사이의 정수 하나 (예, 9)를 입력
  - 이후 키보드로부터 a개의 정수를 입력
  - 입력 받은 정수를 모두 더하여 그 합을 출력

```
void main (int argc, char* argv[]) {
      int i, val, sum=0;
      printf("정수 값 %d개를 입력하세요: ", *argv[1]-'0');

      for (i = 0; i < (*argv[1] - '0'); i++) {
          scanf("%d", &val);
          sum += val;
      }
      printf("전체 합계는 %d입니다.₩n", sum);
}
```

47

➔ 개선할 점은??

실습 문제 (숙박관리 프로그램 확장 2)

- 숙박관리 프로그램 작성 (Version 3)
  - 목적: Chapter 18-2의 프로그램 확장 (구현된 함수를 함수 포인터로 호출하기)
  - 함수 포인터 정의: void (*ptrfunction)( )
  - 함수 포인터를 사용하여 각 구현된 함수를 호출

```
**********************
** 숙박관리 프로그램 **
**********************
1.입실, 2.퇴실, 3.조회, 4.종료:  1
층 번호 입력(1~3층): 1
방 번호 입력(1~5호): 1
고객이름(1자): A


1.입실, 2.퇴실, 3.조회, 4.종료:  3
A       0       0       0       0
0       0       0       0       0
0       0       0       0       0


1.입실, 2.퇴실, 3.조회, 4.종료:
```

실습 문제 (Lab 1)

- 명령어 인자를 사용하여 n!을 계산하는 프로그램 작성
  - 아래와 같이 동작

```
C:₩Users>Day001
Usage:: Day001 [숫자 (1<= 숫자 <= 13)]
C:₩Users>Day001 14
Usage:: Day001 [숫자 (1<= 숫자 <= 13)]
C:₩Users>Day001 13
Usage:: Day001 [숫자 (1<= 숫자 <= 13)]
팩토리얼 결과는 1932053504
```

```
<문자열을 정수로 변환 함수>
int atoi(char *abc)
```

실습 시간 (2019년 10월 15일)

- 예제1 (18-2장): ArrPtrAndPtrArr.c, 2DArrParam.c, 2DArrAccessType.c
- 예제2 (19장): FuntionPointer.c, UsefulFunctionPointer.c, (VoidTypePointer.c, ArgcArgv.c, ArgvParamType.c,) ArgvEndNull.c
- Lab 문제: 숙박관리프로그램 확장 3, Lab1.c
- ➔ **예제 6개, Lab 2문제**