포인터의 포인터

- 더블 포인터와 다중 포인터 -

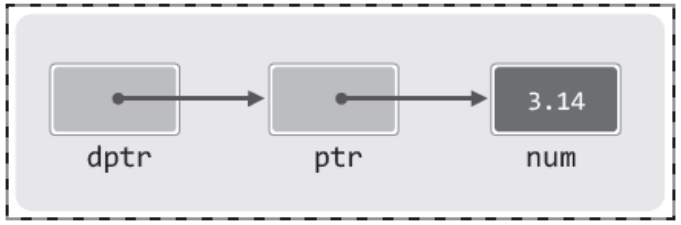
성공회대학교 IT융합자율학부
소프트웨어공학전공
홍 성 준

포인터의 포인터에 대한 이해

◎ 포인터의 포인터

- 포인터 변수를 가리키는(포인터 변수의 주소값을 저장하는) 또 다른 포인터 변수
- 이중 포인터, 더블 포인터
- 포인터 변수 선언에 사용하는 * 연산자를 두 개 이어서 선언

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



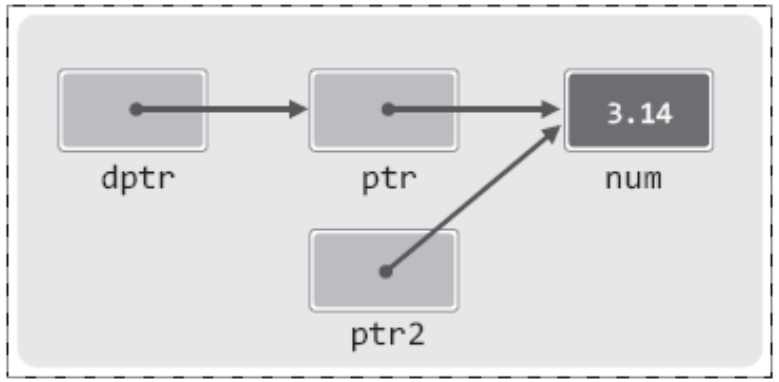
- *dptr 는 포인터 변수 ptr을 의미
- *(*dptr)는 변수 num을 의미하며 괄호를 생략하여 일반적으로 **dptr로 표기

포인터의 포인터에 대한 이해

◎ 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;
}
```




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

- 변수 num에 접근하는 방법? num, *ptr, **dptr, *ptr2

포인터의 포인터에 대한 이해

◎ 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;
}
```

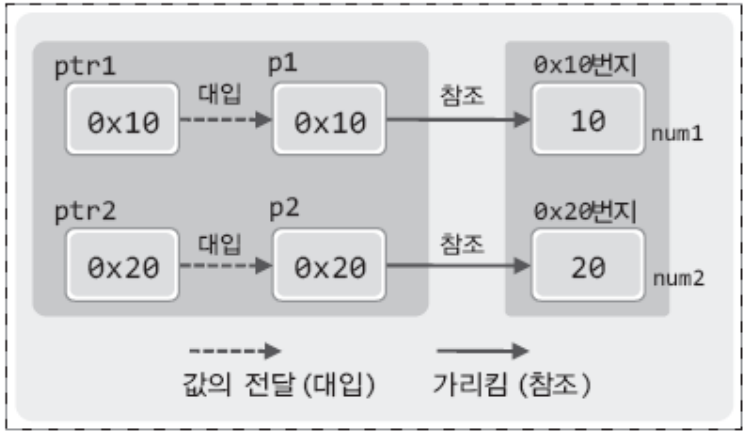
포인터의 포인터에 대한 이해

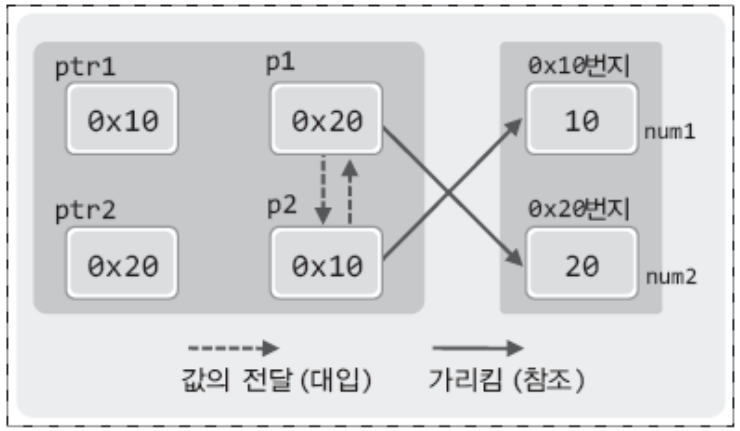
◎ PointerSwapFail.c

```
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;
}
```






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

포인터의 포인터에 대한 이해

◎ 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;
}
```

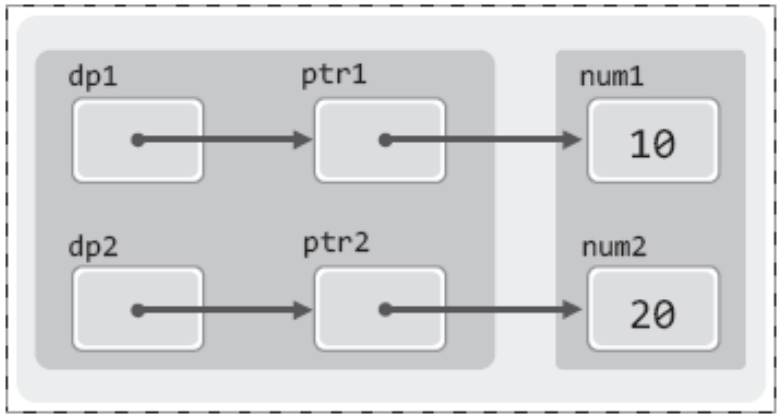
포인터의 포인터에 대한 이해

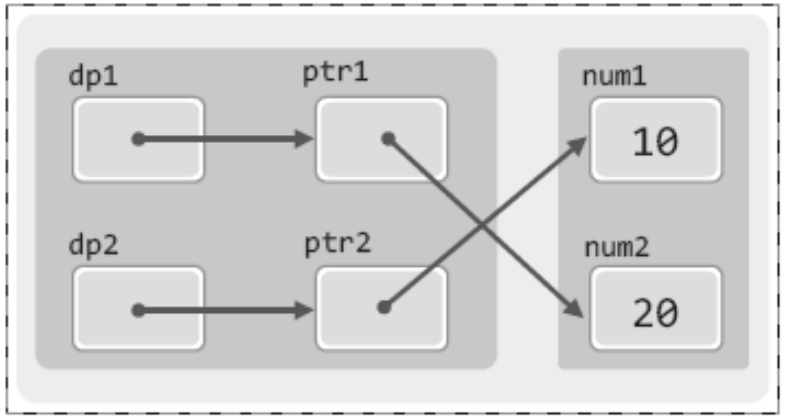
◎ PointerSwapSuccess.c

```
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;
}
```






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

포인터의 포인터에 대한 이해

◎ 배열과 배열 이름의 자료형

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

- 배열 이름 arr1의 포인터 형은 int *, arr2의 포인터형은 double *

◎ 포인터 배열과 포인터 배열 이름의 자료형

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

- 싱글 포인터를 원소로 갖는 포인터 배열에서 배열 이름 arr1의 포인터 형은 int **, arr2의 포인터 형은 double **

포인터의 포인터에 대한 이해

◎ 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
```

다중 포인터 변수

◎ 다중 포인터

- 포인트 변수의 선언에 있어 * 연산자가 둘 이상 사용되어 선언되는 포인터 변수

◎ 삼중 포인터

- 이중 포인터를 가리키는(이중 포인터의 주소 값을 저장하는) 포인터

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

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

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

```
100 100
```

포인터의 필요성

◎ 변수의 지역성(locality)을 극복

- 함수 외부에 선언된 변수에 접근하는 것을 방법을 제시

◎ 연속된 메모리의 참조

- 배열, 구조체와 같이 연속된 메모리를 할당 받는 자료 형에 접근하기 위해서 사용

◎ 자료구조와 알고리즘에서의 활용

- 데이터의 효율적인 표현과 저장 방법을 구현하기 위해 포인터가 필요