

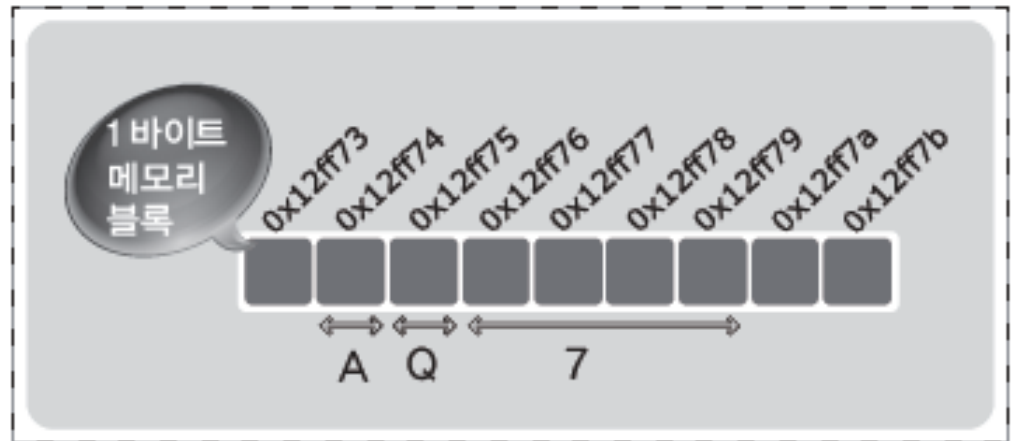
## Chapter 12. 포인터의 이해



## Chapter 12-1. 포인터란 무엇인가?

# 주소 값의 저장을 목적으로 선언되는 포인터 변수

```
int main(void)
{
    char ch1='A', ch2='Q';
    int num=7;
    . . . .
}
```



변수 num이 저장되기 시작한 주소 0x12ff76이 변수 num의 주소 값이다.

이러한 정수 형태의 주소 값을 저장하는 목적으로 선언되는 것이 포인터 변수이다.

# 포인터 변수와 & 연산자 맛보기

“정수 7이 저장된 int형 변수 num을 선언하고 이 변수의 주소 값을 저장할 위한 포인터 변수 pnum을 선언하자. 그리고 나서 pnum에 변수 num의 주소 값을 저장하자.”



코드로 옮긴 결과

```
int main(void)
{
    int num=7;
    int * pnum;
    pnum = &num;
    . . . .
}
```

포인터 변수 pnum의 선언  
num의 주소 값을 pnum에 저장

포인터 변수의 크기는 시스템의 주소 값 크기에 따라서 다르다.

16비트 시스템 → 주소 값 크기 16비트 → 포인터 변수의 크기 16비트!

32비트 시스템 → 주소 값 크기 32비트 → 포인터 변수의 크기 32비트!

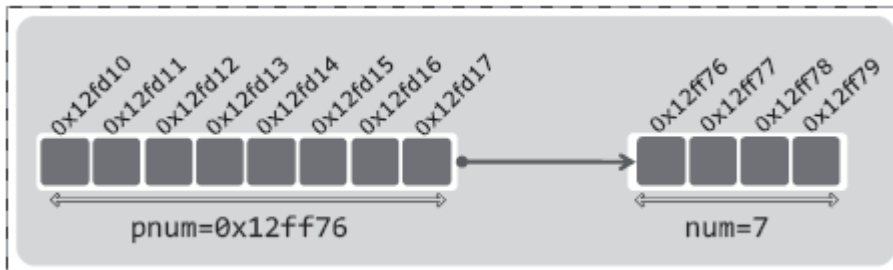
int \* pnum 의 선언에서...

pnum      포인터 변수의 이름

int \*      int형 변수의 주소 값을 저장하는 포인터 변수의 선언



메모리의 저장상태



이 상태를 다음과 같이 표현한다.

포인터 변수 pnum이 변수 num을 가리킨다.

# 포인터 변수 선언하기

가리키고자 하는 변수의 자료형에 따라서 포인터 변수의 선언방법에는 차이가 있다.

포인터 변수에 저장되는 값은 모두 정수로 값의 형태는 모두 동일하지만, 그래도 선언하는 방법에 차이가 있다(차이가 있는 이유는 메모리 접근과 관련이 있다).

```
int * pnum1;
```

int \* 는 int형 변수를 가리키는 pnum1의 선언을 의미함

```
double * pnum2;
```

double \* 는 double형 변수를 가리키는 pnum2의 선언을 의미함

```
unsigned int * pnum3;
```

unsigned int \* 는 unsigned int형 변수를 가리키는 pnum3의 선언을 의미함



일반화

```
type * ptr;
```

type형 변수의 주소 값을 저장하는 포인터 변수 ptr의 선언

# 포인터의 형(Type)

<code>int *</code>	int형 포인터
<code>int * pnum1;</code>	int형 포인터 변수 pnum1
<code>double *</code>	double형 포인터
<code>double * pnum2;</code>	double형 포인터 변수 pnum2



<code>type *</code>	type형 포인터
<code>type * ptr;</code>	type형 포인터 변수 ptr

포인터 변수 선언에서 \* 의 위치에 따른 차이는 없다. 즉, 다음 세 문장은 모두 동일한 포인터 변수의 선언문이다.

<code>int * ptr;</code>	// int형 포인터 변수 ptr의 선언
<code>int* ptr;</code>	// int형 포인터 변수 ptr의 선언
<code>int * ptr;</code>	// int형 포인터 변수 ptr의 선언



## Chapter 12-2. 포인터와 관련 있는 & 연산자와 \* 연산자

```
int main(void)
{
    int num1=100, num2=100;
    int * pnum;

    pnum=&num1;    // 포인터 pnum이 num1을 가리킴
    (*pnum)+=30;    // num1+=30; 과 동일

    pnum=&num2;    // 포인터 pnum이 num2를 가리킴
    (*pnum)-=30;    // num2-=30; 과 동일

    printf("num1:%d, num2:%d \n", num1, num2);
    return 0;
}
```





# 변수의 주소 값을 반환하는 & 연산자

```
int main(void)
{
    int num = 5;
    int * pnum = &num;
    . . . .
}
```

& 연산자는 변수의 주소 값을 반환하므로 상수가 아닌 변수가 피연산자이어야 한다. & 연산자의 반환 값은 포인터 변수에 저장을 한다.

```
int main(void)
{
    int num1 = 5;
    double * pnum1 = &num1; // 일치하지 않음!

    double num2 = 5;
    int * pnum2 = &num2; // 일치하지 않음!
    . . . .
}
```

num1은 int형 변수이므로  
pnum1은 int형 포인터 변수  
이어야 함

num2는 double형 변수이므로  
pnum2는 double형 포인터 변수  
이어야 함.

int형 변수 대상의 & 연산의 반환 값은 int형 포인터 변수에, double형 변수 대상의 & 연산의 반환 값은 double형 포인터 변수에 저장한다.

# 포인터가 가리키는 메모리를 참조하는 \* 연산자

```
int main(void)
```

```
{
```

```
    int num=10;
```

```
    int * pnum=&num;
```

```
    *pnum=20;
```

```
    printf("%d", *pnum);
```

```
    . . . .
```

```
}
```

pnum이 num을 가리킨다.  
pnum이 가리키는 공간(변수)에 20을 저장  
pnum이 가리키는 공간(변수)에 저장된 값 출력

\*pnum은 num을 의미한다.

따라서 num을 놓을 자리에 \*pnum을 놓을 수 있다.

```
int main(void)
```

```
{
```

```
    int num1=100, num2=100;
```

```
    int * pnum;
```

```
    pnum=&num1;    // 포인터 pnum이 num1을 가리킴
```

```
    (*pnum)+=30;    // num1+=30; 과 동일
```

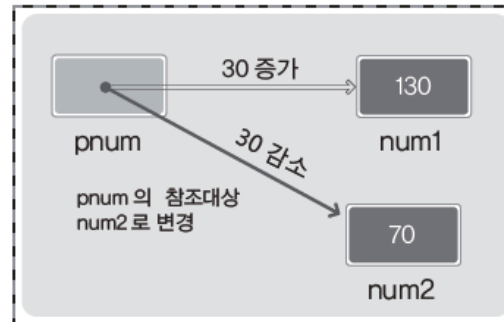
```
    pnum=&num2;    // 포인터 pnum이 num2를 가리킴
```

```
    (*pnum)-=30;    // num2-=30; 과 동일
```

```
    printf("num1:%d, num2:%d \n", num1, num2);
```

```
    return 0;
```

```
}
```



실행결과

num1:130, num2:70

## 실습 12-2

아래의 예제 실행 시 변수와 포인터 변수의 관계를 그림을 그려서 설명해보자. 또한 출력의 결과도 예상해보자.

```
int main(void)
{
    int num=10;
    int * ptr1=&num;
    int * ptr2=ptr1;

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



## 실습 12-3

```
01 // file: dereference.c
02 #include <stdio.h>
03
04 int main(void)
05 {
06     int data = 100;
07     char ch = 'A';
08     int *ptring = &data;
09     char *ptrchar = &ch;
10     printf("간접참조 출력: %d %c\n", *ptring, *ptrchar);
11
12     *ptring = 200; //변수 data를 *ptring로 간접참조하여 그 내용을 수정
13     *ptrchar = 'B'; //변수 ch를 *ptrchar로 간접참조하여 그 내용을 수정
14     printf("직접참조 출력: %d %c\n", data, ch);
15
16     return 0;
17 }
```

변수 data와 같다.

변수 ch와 같다.



# 다양한 포인터 형이 존재하는 이유

---

포인터 형은 메모리 공간을 참조하는 방법의 힌트가 된다. 다양한 포인터 형을 정의한 이유는 \* 연산을 통한 메모리의 접근기준을 마련하기 위함이다.

**int형 포인터 변수로 \* 연산을 통해 메모리(변수) 접근 시**

**4바이트 메모리 공간에 부호 있는 정수의 형태로 데이터를 읽고 쓴다.**

.

**double형 포인터 변수로 \* 연산을 통해 메모리(변수) 접근 시**

**8바이트 메모리 공간에 부호 있는 실수의 형태로 데이터를 읽고 쓴다.**



```
int main(void)
{
    double num=3.14;
    int * pnum=&num;
    printf("%d", *pnum);
    . . . .
}
```

형 불일치! 컴파일은 된다.  
pnum이 가리키는 것은 double형 변수인데, pnum이 int형 포인터 변수이므로 int형 데이터처럼 해석!.

주소 값이 정수임에도 불구하고 int형 변수에 저장하지 않는 이유는 int형 변수에 저장하면 메모리 공간의 접근을 위한 \* 연산이 불가능하기 때문이다.



# 잘못된 포인터의 사용과 널 포인터

```
int main(void)
{
    int * ptr;
    *ptr=200;
    . . . .
}
```

위험한 코드

ptr이 쓰레기 값으로 초기화 된다. 따라서 200이 저장되는 위치는 어디인지 알 수 없다! 매우 위험한 행동!

```
int main(void)
{
    int * ptr=125;
    *ptr=10;
    . . . .
}
```

위험한 코드

포인터 변수에 125를 저장했는데 이곳이 어디인가? 역시 매우 위험한 행동!

```
int main(void)
{
    int * ptr1=0;
    int * ptr2=NULL;
    . . . .
}
```

안전한 코드

잘못된 포인터 연산을 막기 위해서 특정한 값으로 초기화하지 않는 경우에는 **널 포인터**로 초기화하는 것이 안전하다.

**널 포인터 NULL**은 숫자 0을 의미한다. 그리고 0은 0번지를 뜻하는 것이 아니라, 아무것도 가리키지 않는다는 의미로 해석이 된다.

## 실습 12-4

```
01 // file: nullpointer.c
02 #include <stdio.h>
03
04 int main(void)
05 {
06     int *ptr1, *ptr2, data = 10;
07     ptr1 = NULL;
08
09     printf("%p\n", ptr1);
10     //printf("%p\n", ptr2); ← 초기값이 없어서 컴파일 오류발생
11     printf("%d\n", data);
12
13     return 0;
14 }
```





## 실습 12-5

자료형 char, int, double의 변수를 각각 선언하여 적당한 값을 저장하고, 다시 그 변수의 주소를 저장하기 위해 자료형 char, int, double의 포인터 변수를 각각 선언해 그 주소값과 저장한 후, 주소값과 저장값을 출력하는 프로그램을 작성해보자.

- char 포인터 변수 선언: char \*pc
- int 포인터 변수 선언: int \*pm
- double 포인터 변수 선언: double \*px

변수명	주소값	저장값
c	002DFE0B	@
m	002DFDF0	100
x	002DFDD4	5.830000

### basicpointer.c

```
01 // file: basicpointer.c
02 #include <stdio.h>
03
04 int main(void)
05 {
06     char c = '@';
07     char *pc = ____;
08     int m = 100;
09     int *pm = ____;
10     double x = 5.83;
11     double *px = ____;
12
13     printf("변수명   주소값   저장값\n");
14     printf("-----\n");
15     printf("%3s %12p %9c\n", "c", ____, c);
16     printf("%3s %12p %9d\n", "m", ____, m);
17     printf("%3s %12p %9f\n", "x", ____, x);
18
19     return 0;
20 }
```

## 실습 12-6

int형 변수 num1과 num2를 선언과 동시에 각각 10과 20으로 초기화하고, int형 포인터 변수 ptr1과 ptr2를 선언하여 각각 num1과 num2를 가리키게 하자. 그리고 이 상태에서 포인터 변수 ptr1과 ptr2를 이용해서 num1의 값을 10 증가시키고, num2의 값을 10 감소시키자. 이제 두 포인터 변수 ptr1과 ptr2가 가리키는 대상을 서로 바꾸자. 즉, 포인터 변수 ptr1이 num2를 가리키게 하고, 포인터 변수 ptr2가 num1을 가리키게 하자. 그리고 마지막으로 ptr1과 ptr2가 가리키는 변수에 저장된 값을 출력하자.

