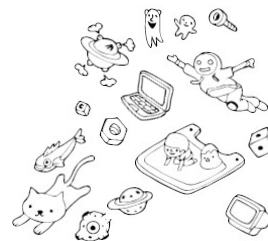
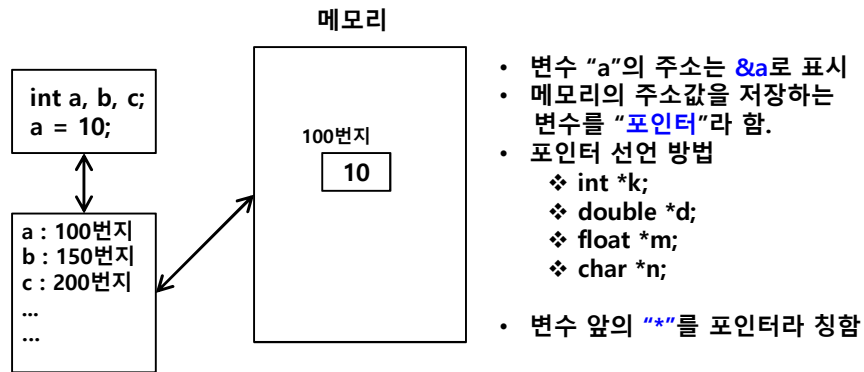


Chapter 12. 포인터의 이해



Chapter 12-1. 포인터란 무엇인가?

변수의 이름과 주소



주소를 저장할 변수

`int a = 10;` // 정수 방 a를 만들고 10을 저장해라!
`double b = 23.4;` // 실수 값 저장할 방 b를 만들고 23.4를 저장해라!
`char c = 'k';` // 하나의 문자를 저장할 방 c를 만들고 'k'를 저장해라!

주소를 저장할 방을 만들려면....

`int* p = &a;` // 정수 방의 주소를 저장할 주소 방 p를 만들고 &a를 저장해라!
`double* q = &b;` // 실수 방의 주소를 저장할 방 q를 만들고 &b를 저장해라!
`char* r = &c;` // 문자 방의 주소를 저장할 방 r를 만들고 &c를 저장해라!

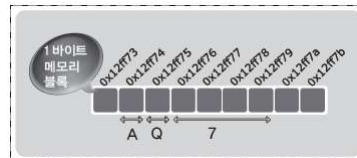
p방에는 a 방의 주소가 저장되어 있으므로 아래 3개의 명령은 모두 동일하다!!

```
a = 100;
*(&a) = 100;
*p = 100;
```

- `int *p = &a;` // 포인터 변수 p를 선언하고 a의 주소 값 저장
- `*p = 100;` // 포인터 변수 p가 가리키는 메모리 공간에 100을 저장

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.
- 변수들의 주소 값을 저장하는 목적으로 선언되는 것이 "포인터 변수"이다.
- 주소 값은 "정수 값" 이다!!!

<포인터 변수 선언 및 할당 방법>

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



포인터 변수와 & 연산자 맛보기

"정수 7이 저장된 int형 변수 num을 선언하고 이 변수의 주소 값을 저장할 수 있는 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 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!

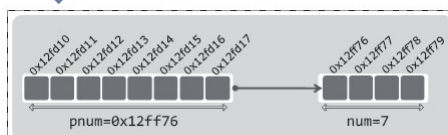
포인터 변수 pnum의 선언
pnum = # num의 주소 값을 pnum에 저장

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 **정수 값**의 형태로 모두 동일하지만, 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)

int *	int형 포인터
int * pnum1;	int형 포인터 변수 pnum1
double *	double형 포인터
double * pnum2;	double형 포인터 변수 pnum2

↓ 일반화

type *	type형 포인터
type * ptr;	type형 포인터 변수 ptr

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

```
int * ptr;      // int형 포인터 변수 ptr의 선언
int* ptr;      // int형 포인터 변수 ptr의 선언
int * ptr;      // int형 포인터 변수 ptr의 선언
```



Chapter 12-2. 포인터와 관련 있는 & 연산자와 * 연산자

변수의 주소 값을 반환하는 & 연산자

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

- & 연산자는 변수의 주소 값을 반환하므로 상수가 아닌 **변수**가 피연산자이어야 한다.
- & 연산자의 반환 값은 **포인터 변수**에 저장한다.

```
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 변수이어야 함

num2는 double형 변수이므로 pnum2는 double형 포인터 변수이어야 함.

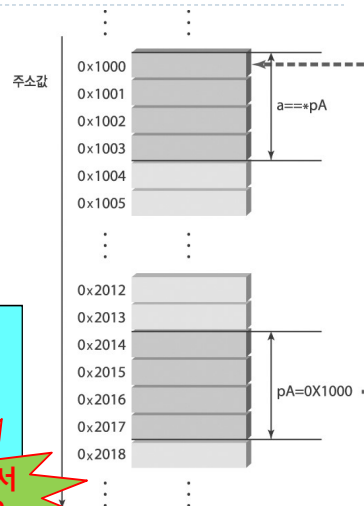
- int형 변수 대상의 & 연산의 반환 값은 int형 포인터 변수에,
- double형 변수 대상의 & 연산의 반환 값은 double형 포인터 변수에 저장한다.

변수의 주소 값을 반환하는 & 연산자

```
int main (void) {
    int a = 2005;
    int * pA = &a;
    printf("%d\n", a);    // 직접 접근
    printf("%d\n", *pA); // 간접 접근
    printf("%p\n", pA);  // a 방 주소
    . . . . .
}
```

주소 저장할 방 선언 : `int * pA;`
 주소 방에 주소 저장 : `pA = &a;`
 pA에는 a의 주소 저장 (`pA == &a`)
 pA는 a의 주소인 &a와 동일
 *pA는 a와 동일 (`*pA == a`)

포인터에서
가장 중요



포인터가 가리키는 메모리를 참조하는 * 연산자

```
int main(void)
{
    int num=10;    pnum이 num을 가리킨다.
    int * pnum=&num;
    *pnum=20;    pnum이 가리키는 공간(변수)에 20을 저장
    printf("%d", *pnum);
    . . . . .    pnum이 가리키는 공간(변수)에 저장된 값 출력
}
```

- *pnum은 num을 의미한다.
- 따라서 num을 놓을 자리에 *pnum을 놓을 수 있다.

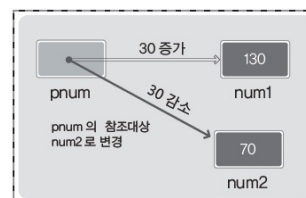
```
int main(void)
{
    int num1=100, num2=100;
    int * pnum;

    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;
}
```

PointerOperation.c



실행결과

num1:130, num2:70

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 변수**로 * 연산을 통해 메모리(변수) 접근 시 **4바이트 메모리 공간**에 부호 있는 **정수의 형태로** 데이터를 읽고 쓴다.
- **double형 포인터 변수**로 * 연산을 통해 메모리(변수) 접근 시 **8바이트 메모리 공간**에 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 때문**



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!**

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

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

위험한 코드

```
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 NULL은 숫자 0을 의미한다. 그리고 0은 0번지를 뜻하는 것이 아니라, **“아무것도 가리키지 않는다”**는 의미로 해석이 된다.



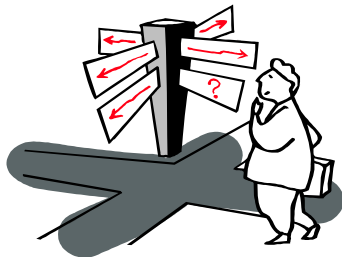
실습문제 (Lab1)

▶ 아래와 같은 예제를 작성해보라.

- ▶ 정수형 변수 num1과 num2를 선언과 동시에 각각 10과 50으로 초기화
- ▶ 정수형 포인터 변수 ptr1과 ptr2를 선언하고 각각 num1과 num2를 가리키게 함
- ▶ 포인터 변수를 사용하여 num1 값을 10 증가시키고, num2 값을 10 감소시킴
- ▶ 두 포인터 변수가 가리키는 대상을 서로 바꿈
- ▶ 두 포인터 변수가 가리키는 변수에 저장된 값을 출력

```
int num1 = 10, num2 = 50;
int *ptr1, *ptr2;

// ptr1, ptr2가 num1, num2를 가리키게 함
// ptr1, ptr2를 사용하여 num1, num2 값을 10씩 변경 (증가/감소)
// ptr1, ptr2가 가리키는 대상을 서로 바꿈
// ptr1, ptr2가 가리키는 변수 값을 출력
```



Chapter 12가 끝났습니다. 질문 있으신지요?