4. 포인터

*: 애스터리스크

포인터 (pointer) : 메모리의 주소를 가지고 있는 변수 (X 메모리에는 각 바이트마다 고유한 주소가 매겨짐) 바이트단위 썬프레소

- 프로그램에서 변수를 만들면 변수는 컴파일러에 의해 메모리 공간의 비어있는 위치 차지함.
- 변수의 크기에 따라 달라지는 차지하는 메모리 공간
  - char형 변수 : 1바이트
  - int형 변수 : 4바이트
  - float형 변수 : 4바이트

주소 (&) 연산자 : 변수의 이름을 받아서 변수의 주소 반환

1. 포인터의 선언

포인터 : 변수의 주소를 가지고있는 변수

int *p

정수를 가리키는 포인터 p

쉽게말하면!
- p = &i ⇒ 이때의 p는 i의 주소!!
- *p ⇒ p가 가리키고 있는 주소 가리고 있는 값

ex)
```
int i=0; // 정수 변수 i가 선언되고, 10으로 초기화됨.
int *p;  // 정수 포인터가 선언된다.
p=&i;    // 포인터 p에 변수 i의 주소를 저장한다.
```

ex)
```
int i=0;
double f=12.3;
int *pi=NULL;
double *pf=NULL;

pi=&i;
pf=&f;
printf("%u %u\n", pi, &i);
printf("%u %u\n", pf, &f);
return 0;
```

정수를 가리키는 포인터: 지역변수로 포인터를 선언하고 초기화를 시켜지 않으면 쓰레기값을 갖게된다. 이때는 NULL 값을 저장하며 놀는 것이 좋다. NULL은 주소 0이다

double형 실수를 가리키는 포인터

포인터 pi에 변수 i의 주소를 대입한다.

ex)
```
int i=3000;
int *p=NULL; → 포인터의 선언

p=&i; → 변수 i의 주소를 p에 대입

printf("p= %u\n", p);
printf("&i= %u\n", &i); ] → p가 가리키는 주소에서 데이터를 읽어온다

printf("i= %d\n", i);
printf("*p= %d\n", *p); ] → 포인터를 통한 간접참조

return 0;
```

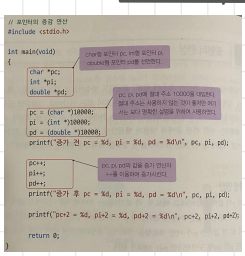
※ 포인터 사용시 주의할점

1. 초기화하지 않고 사용함 → 쓰레기값
2. NULL 포인터의 사용
   → 포인터가 아무것도 가리키고 있지 않을 때 : NULL(0) [int *p=NULL;]
3. 포인터타입 = 변수의 타입
4. 절대주소사용 → 안좋음

```
if (p!=NULL)
    printf("포인터p는 유효합니다");
else
    printf("포인터p는 유효하지않습니다");
```
← 포인터가 유효한지 아닌지 검사할때 →

포인터연산

포인터타입	++연산후 증가되는값
char	1
short	2
int	4
float	4
double	8



실행결과
증가전 pc=10000, pi=10000, pd=10000
증가후 pc=10001, pi=10004, pd=10008
pc+2=10003, pi+2=10012, pd+2=10024

간접참조연산자와 증감연산자

*p++; ⇒ p가 가리키는 위치에서 값을 가져온후 포인터 p를 증가한다
(*p)++; ⇒ 포인터가 가리키는 대상의 값을 증가한다.

```
int i=10;
int *pi=&i;
(*pi)++ ⇒ i=11, pi=0012FF60
*pi++   ⇒ i=10, pi=0012FF64
```
} int의 크기만큼 증가

V=*p++ : p가 가리키는 값을 v에 대입한 후에 p를 증가한다.
V=(*p)++ : p가 가리키는 값을 v에 대입한 후에 v에 대입한 후에 p가 가리키는 값을 증가한다.
V=*++p : p를 증가시킨 후에 p가 가리키는 값을 v에 대입한다.
V=++*p : p가 가리키는 값을 가져온 후에 그값을 증가하여 v에 대입한다

포인터의 형변환

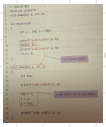
```
double *pd=&f;
int *pi;

pi=(int *)pd; double형 포인터를 int형 포인터로 변환
```

ex)
```
int data=0x0A0B0C0D;
char *pc;

pc=(char*)&data;

for(int i=0; i<4; i++) {
    printf("*(pc+%d)=%02X\n", i, *(pc+i));
}
return 0;
```

실행결과
```
*(pc+0)=0D
*(pc+1)=0C
*(pc+2)=0B
*(pc+3)=0A
```

포인터와 함수

1) 함수호출시 인수전달방식
- 값에 의한 호출 (call-by-value) : 복사본이 전달된다.
- 참조에 의한 호출 (call-by-reference) : 원본이 전달된다.



실행결과
```
a=100 b=200
x=100 y=200
x=200 y=100
a=100 b=200
```
⇒ swap()안에서는 변수 x와 y의 값이 교환되었지만 main()안에서는 변수 a와 b의 값이 교환되지 않음
↳ 이유: "값에 의한 호출이기 때문" ⇒ 함수의 인수로 변수의 값만 전달되기 때문에 원본변수 자체를 변경할수없음.

```
int main(void)
{
    int a=100, b=200;
    printf("a=%d b=%d\n", a, b);
    swap(&a, &b); // 변수의 값이 아니라 변수의 주소를 전달한다.
    printf("a=%d b=%d\n", a, b);

    return 0;
}

void swap(int *px, int *py) // px는 &a의 값을 가지고있고, *px의 값은 a의 값이 된다.
{
    int tmp;
    tmp=*px; // 결과적으로 tmp=a;와 같은 의미
    *px=*py; // 결과적으로 a=b;와 같은 의미
    *py=tmp; // 결과적으로 b=tmp;와 같은 의미
}
```

실행결과
```
a=100 b=200
a=200 b=100
```
올바르게 교환됨

···→ 변수의 주소가 전달된다.

scanf() 함수 : '참조에 의한 호출'을 사용하는 전형적인 예

TIP 함수가 포인터를 통해 값을 변경할 수 없게 하려면?

```
void sub(const int *p)
{
    *p=0; // 오류!! const로 선언되면 이게해변수 p를 통해 값을 변경할수 없음
}
```

∴ 일반적으로 어떤 계산이나 동작을 위해 값만을 필요로 한다면 ? 값에 의한 호출
함수가 외부에서 선언된 변수의 값을 변경할 필요가 있다면? 참조에 의한 호출

4. 포인터

포인터와 배열의 관계

```
int a[] = {10,20,30,40,50};
printf("&a[0]=%u \n", &a[0]);
printf("&a[1]=%u \n", &a[1]);
printf("&a[2]=%u \n", &a[2]);
printf("a=%u\n", a);
```

배열의 첫번째 요소 = a[0] → 주소: &a[0]
같은 식으로 두번째, 세번째 주소값은 각각 &a[1], &a[2]
이 주소값들을 부호없는 정수값으로 출력해보다

출력결과
&a[0]=1245008
&a[1]=1245012
&a[2]=1245016
a=1245008
→ 완전히 동일함

배열이름을 부호없는 정수값으로 출력
첫번째 요소의 주소값과 완전히 동일함

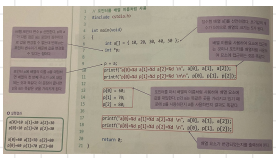
배열이름 = 배열을 가리키는 포인터
배열의 이름을 정수형식으로 출력하면 배열의 첫 번째 요소의 주소와 같다.

```
{
  int a[]= {10,20,30,40,50};
  printf("a=%u\n", a); //배열의이름을 포인터처럼 사용할수있다.
  printf("a+1=%u\n", a+1); // a+1의 값 출력 / a+1은 a보다 4가 크며,
                           a[1]의 주소와 같다.
  printf("*a=%d\n", *a); // 배열의 이름이 포인터라고 했으므로 배열의 이름이
                         가리키는 곳의 내용 *a를 출력하면 a[0]와 같다.
  printf("*(a+1)=%d\n",*(a+1)); //*(a+1)은 a[1]과 같다.
  return 0;
}
```

실행결과 a=1245008
a+1=1245012
*a=10
*(a+1)=20

a+i = (배열의시작주소) + (i * 배열요소의 크기) = &a[i]

```
int a[]={10,20,30,40,50};
++a; //컴파일오류: a는 포인터 상수이므로 변경불가
```



배열매개변수

```
void sub(int x)
{
}
```
x에 실제로 가져온 주소가 할당됨

```
void sub(int b[], int size)
{
}
```
실제로 배열 b가 생성되지 않는다.
b는 배열을 가리키는 포인터로서 외부에서 전달된 배열의 주소가 전달됨

```
int main(void)
{
    int a[3]={1,2,3};
    sub(a,3);
}
```
↳ 배열의 이름은 포인터이다.

```
void sub(int b[], int size)
{
  *b=4;
  *(b+1)=5;
  *(b+2)=6;
}
```
배열을 가리키는 포인터
→ 포인터 b를 통하여 원본 배열을 변경할수있음

포인터 사용의 장점

1) 포인터를 이용하면 연결리스트나 이진트리 등의 향상된 자료구조를 만들 수 있음
2) 메모리 메핑 하드웨어 : 메모리처럼 접근할수있는 하드웨어 장치
3) 참조에 의한 호출
4) 동적 메모리 할당

<매개변수로 배열 선언할때>

// 배열매개변수

```
void sub(int b[], int size)
{
    b[0]=4;
    b[1]=5;
    b[2]=6;
}
```
↳ 배열의 이름과 포인터는 근본적으로 같다

// 포인터매개변수

```
Void sub(int *b, int size)
{
    b[0]=4;
    b[1]=5;
    b[2]=6;
}
```
→ 배열표기법을 사용하여 배열에 접근