

3_3장 포인트



이번 장에서 학습할 내용

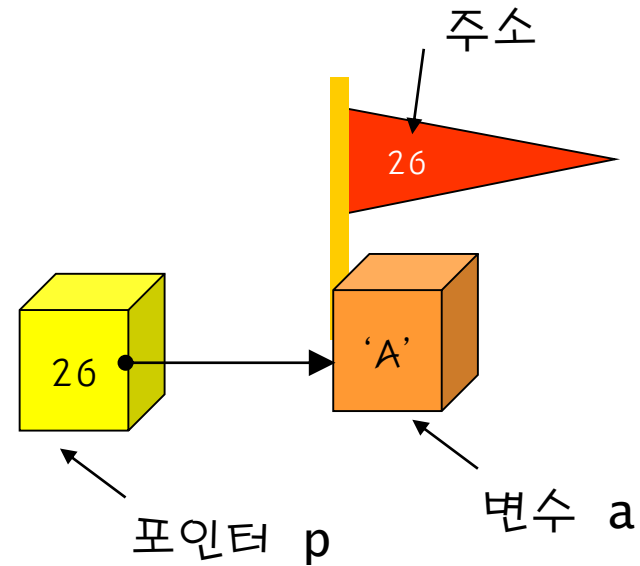


- 포인터이란?
- 변수의 주소
- 포인터의 선언
- 간접 참조 연산자
- 포인터 연산
- 포인터와 배열
- 포인터와 함수

포인터(pointer)

- 포인터: 다른 변수의 주소를 가지고 있는 변수

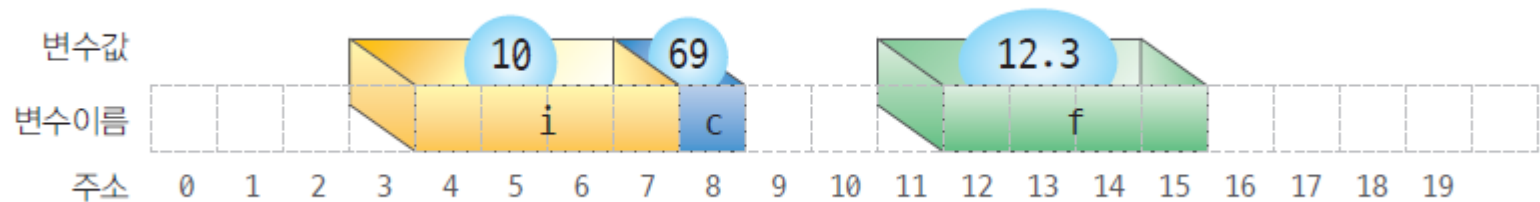
```
char a='A';  
char *p;  
p = &a;
```



변수와 메모리

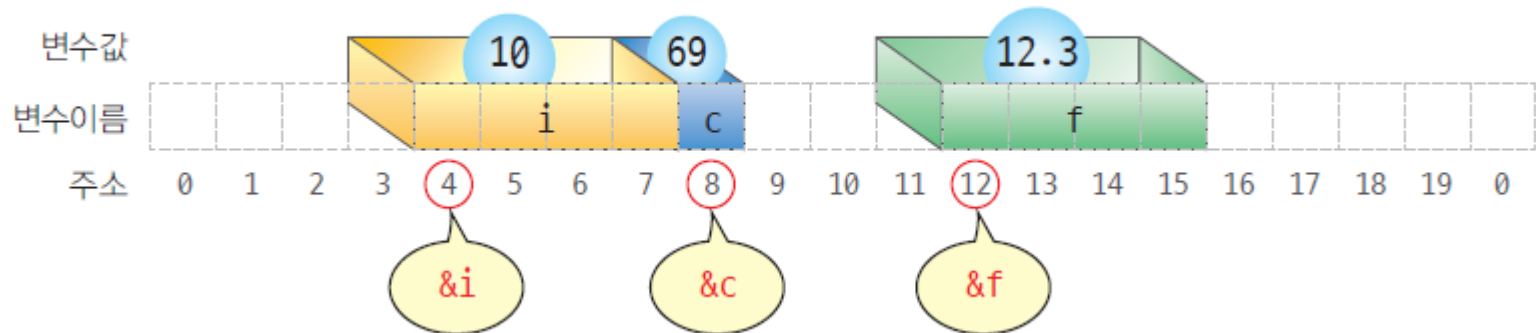
- 변수의 크기에 따라서 차지하는 메모리 공간이 달라진다.
- char형 변수: 1바이트, int형 변수: 4바이트,...

```
int main(void)
{
    int i = 10;
    char c = 69;
    float f = 12.3;
}
```



변수의 주소

- 변수의 주소를 계산하는 연산자: &
- 변수 i의 주소: &i





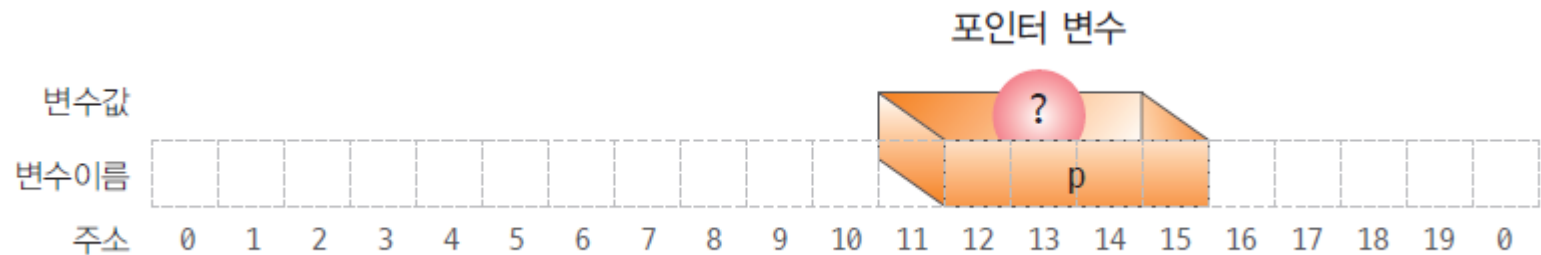
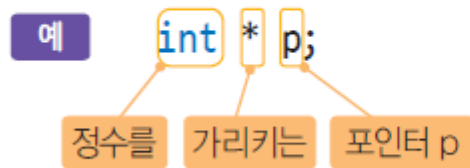
```
1      #include<stdio.h>
2
3      int main(void)
4      {
5          int i = 10;
6          char c = 69;
7          float f = 12.3;
8
9          printf("i 값 = %u , i의 주소 = %u \n", i, &i);
10         printf("i 값 = %d , i의 주소 = %d \n", i, &i);
11
12
13         printf("c 값 = %u , c의 주소 = %u \n", c, &c);
14         printf("c 값 = %d , c의 주소 = %d \n", c, &c);
15         printf("c 값 = %c , c의 주소 = %d \n", c, &c);
16
17         printf("f 값 = %.1f , f의 주소 = %u \n", f, &f);
18         printf("f 값 = %.1f , f의 주소 = %d \n", f, &f);
19
20         return 0;
21     }
```



포인터의 선언

- 포인터 변수 : 변수의 주소를 가지고 있는 변수

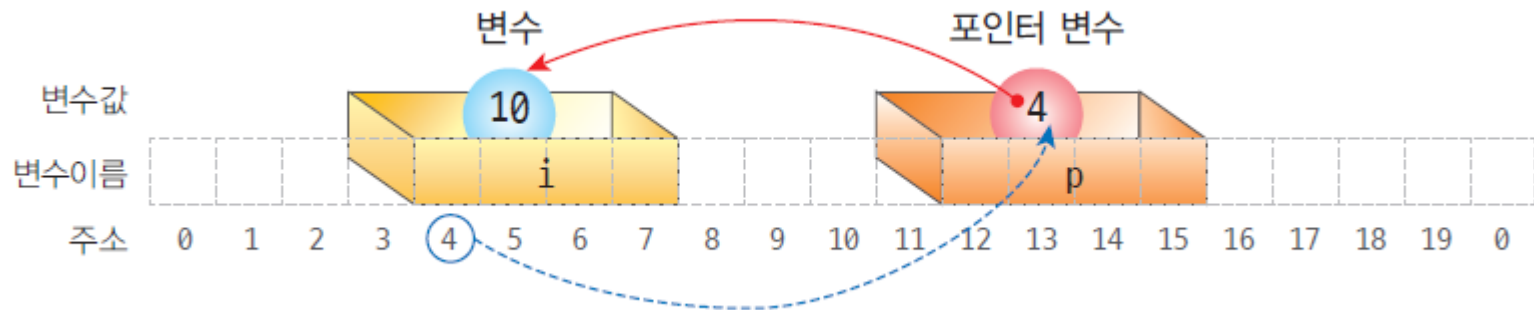
Syntax: 포인터 선언





포인터와 변수의 연결

```
int i = 10;           // 정수형 변수 i 선언
int *p;               // 포인터 변수 p 선언
p = &i;               // 변수 i의 주소가 포인터 p로 대입
```





다양한 포인터의 선언

char c = 'A'; // 문자형 변수 **c**

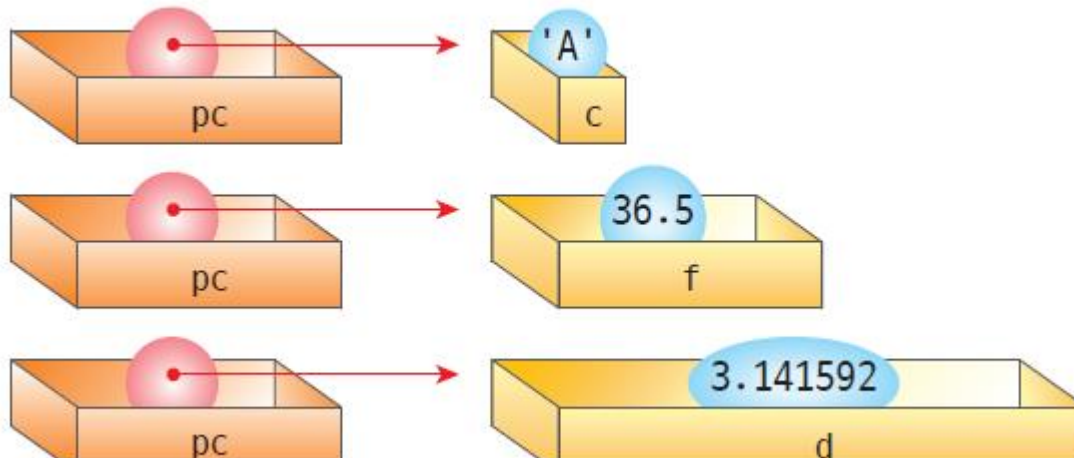
float f = 36.5; // 실수형 변수 **f**

double d = 3.141592; // 실수형 변수 **d**

char *pc = &c; // 문자를 가리키는 포인터 **pc**

float *pf = &f; // 실수를 가리키는 포인터 **pf**

double *pd = &d; // 실수를 가리키는 포인터 **pd**





```
1 #include<stdio.h>
```

```
2  
3 int main(void)
```

```
{
```

```
4  
5     int i = 10; int* pi = &i;
```

```
6     char c = 69; char* pc = &c;
```

```
7     float f = 12.3; float* pf = &f;
```

```
8  
9     printf("i 값 = %u , i의 주소 = %u \n", i, pi);
```

```
10    printf("i 값 = %d , i의 주소 = %d \n", i, pi);
```

```
11  
12    printf("c 값 = %u , c의 주소 = %u \n", c, pc);
```

```
13    printf("c 값 = %d , c의 주소 = %d \n", c, pc);
```

```
14    printf("c 값 = %c , c의 주소 = %d \n", c, pc);
```

```
15  
16    printf("f 값 = %.1f , f의 주소 = %u \n", f, pf);
```

```
17    printf("c 값 = %.1f , f의 주소 = %d \n", f, pf);
```

```
18  
19    return 0;
```

```
20 }
```



```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int i = 10;
```

```
    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;
```

```
}
```



- `#define NULL ((void *)0)`
- 0번지는 일반적으로는 사용할 수 없다(**CPU가 인터럽트를 위하여 사용한다**). 따라서 포인터 변수의 값이 0이면 아무 것도 가리키고 있지 않다고 판단할 수 있다.



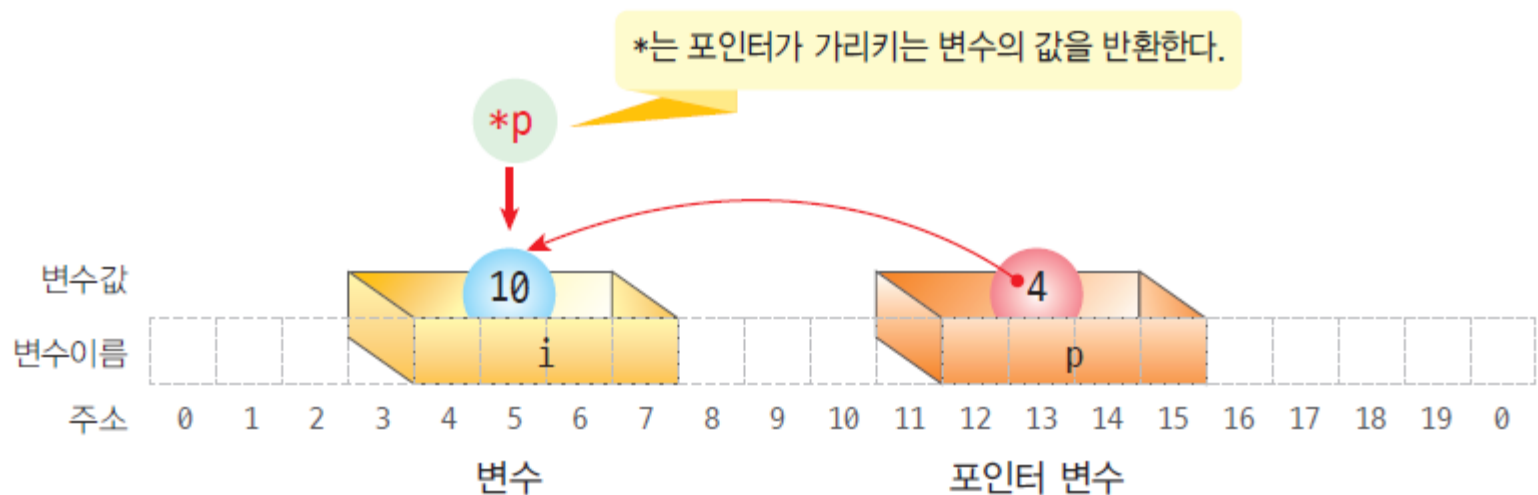
간접 참조 연산자

- 간접 참조 연산자 *: 포인터가 가리키는 값을 가져오는 연산자

```
int i = 10;
```

```
int* p;  
p = &i;
```

```
printf("%d \n", *p);
```

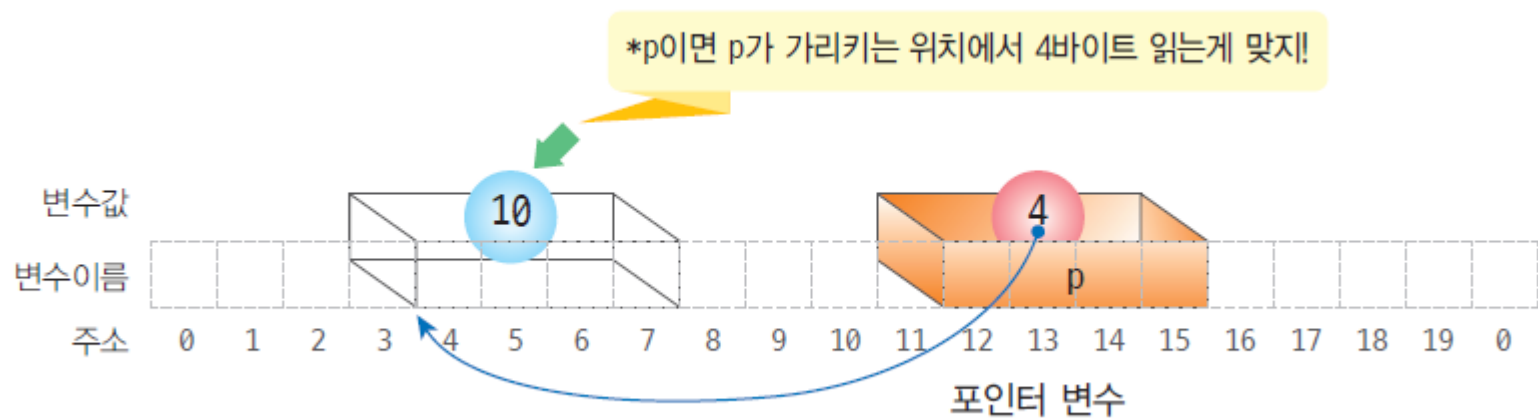




간접 참조 연산자의 해석

- 간접 참조 연산자: 지정된 위치에서 포인터의 타입에 따라 값을 읽어 들인다.

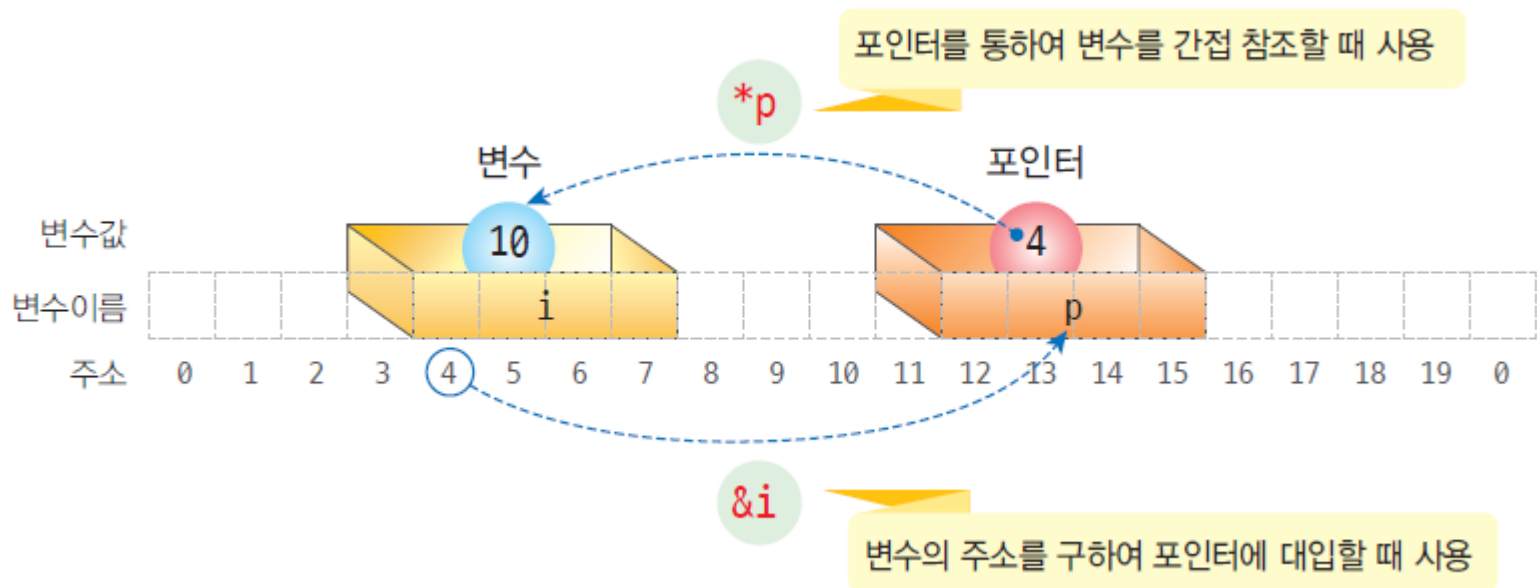
```
int *pi = (int *)10000;  
char *pc = (char *)10000;  
double *pd = (double *)10000;
```





& 연산자와 * 연산자

- & 연산자: 변수의 주소를 반환한다
- * 연산자: 포인터가 가리키는 곳의 내용을 반환한다.





```
1 #include<stdio.h>
2
3 int main(void)
4 {
5     int i = 10; int* pi = &i;
6     char c = 69; char* pc = &c;
7     float f = 12.3; float* pf = &f;
8
9     printf("i 값 = %u , i의 주소 = %u , 주소의 내용 = %d\n", i, pi, *pi);
10    printf("i 값 = %d , i의 주소 = %d\n", i, pi);
11
12    printf("c 값 = %u , c의 주소 = %u , 주소의 내용 = %d\n", c, pc, *pc);
13    printf("c 값 = %d , c의 주소 = %d , 주소의 내용 = %c\n", c, pc, *pc);
14    printf("c 값 = %c , c의 주소 = %d\n", c, pc);
15
16    printf("f 값 = %.1f , f의 주소 = %u , 주소의 내용 = %.1f\n", f, pf, *pf);
17    printf("f 값 = %.1f , f의 주소 = %d\n", f, pf);
18
19    return 0;
20 }
```




포인터 예제

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int x=10, y=20;
```

```
    int *p;
```

```
    p = &x;
```

```
    printf("p = %d\n", p);
```

```
    printf("*p = %d\n\n", *p);
```

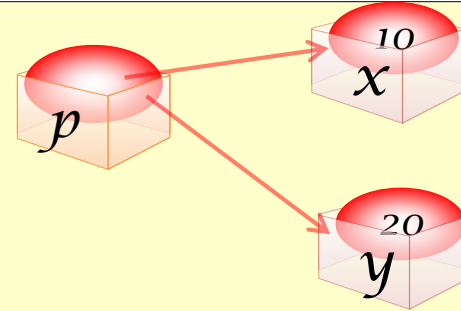
```
    p = &y;
```

```
    printf("p = %d\n", p);
```

```
    printf("*p = %d\n", *p);
```

```
    return 0;
```

```
}
```





포인터 예제 #3

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int i=10;
```

```
    int *p;
```

```
    p = &i;
```

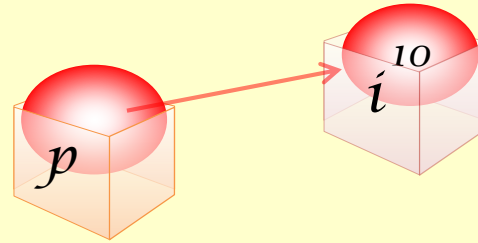
```
    printf("i = %d\n", i);
```

```
    *p = 20;
```

```
    printf("i = %d\n", i);
```

```
    return 0;
```

```
}
```



포인터를 통하여 변수의 값을 변경한다.



```
1  #include<stdio.h>
2
3  int main(void)
4  {
5      int i = 10; int* pi = &i;
6      printf("i 값 = %u , i의 주소 = %u , 주소의 내용 = %d\n", i, pi, *pi);
7
8      *pi = 20;
9      printf("i 값 = %u , i의 주소 = %u , 주소의 내용 = %d\n", i, pi, *pi);
10
11     return 0;
12 }
```



두 값을 교환하는 프로그램 예제 1 (call by value)

```
1  #include<stdio.h>
2  void swap(int a, int b)
3  {
4      int tmp;
5      tmp = a;
6      a = b;
7      b = tmp;
8
9      printf("swap 함수에서: a = %d, b = %d \n", a, b);
10     return 0;
11 }
12 int main(void)
13 {
14     int a = 1, b = 2;
15     printf("swap 호출전: a = %d, b = %d \n", a, b);
16     swap(a, b);
17     printf("swap 호출후: a = %d, b = %d \n", a, b);
18     return 0;
19 }
```



두 값을 교환하는 프로그램 예제 1 (call by reference)

```
1  #include<stdio.h>
2  void swap(int *a, int *b)
3  {
4      int tmp;
5      tmp = *a;
6      *a = *b;
7      *b = tmp;
8
9      printf("swap 함수에서: a = %d, b = %d \n", *a, *b);
10     return 0;
11 }
12 int main(void)
13 {
14     int a = 1, b = 2;
15     printf("swap 호출전: a = %d, b = %d \n", a, b);
16     swap(&a, &b);
17     printf("swap 호출후: a = %d, b = %d \n", a, b);
18     return 0;
19 }
```



포인터 사용시 주의점

- 초기화가 안된 포인터를 사용하면 안된다.

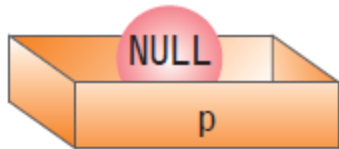
```
int main(void)
{
    int *p;           // 포인터 p는 초기화가 안되어 있음
    *p = 100;         // 위험한 코드
    return 0;
}
```

포인터 사용시 주의점

- 포인터가 아무것도 가리키고 있지 않는 경우에는 **NULL**로 초기화
- **NULL** 포인터를 가지고 간접 참조하면 하드웨어로 감지할 수 있다.

```
int *p = NULL;
```

포인터가 아무것도 가리키지 않을 때는
반드시 NULL로 설정하세요.





포인터 사용시 주의점

- 포인터의 타입과 변수의 타입은 일치하여야 한다.

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int i;
```

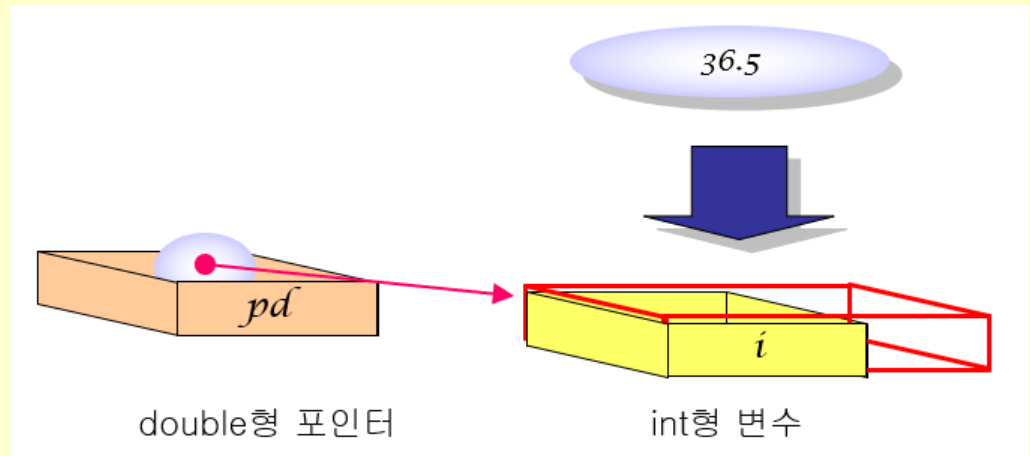
```
    double *pd;
```

```
    pd = &i;           // 오류!
```

```
    *pd = 36.5;
```

```
    return 0;
```

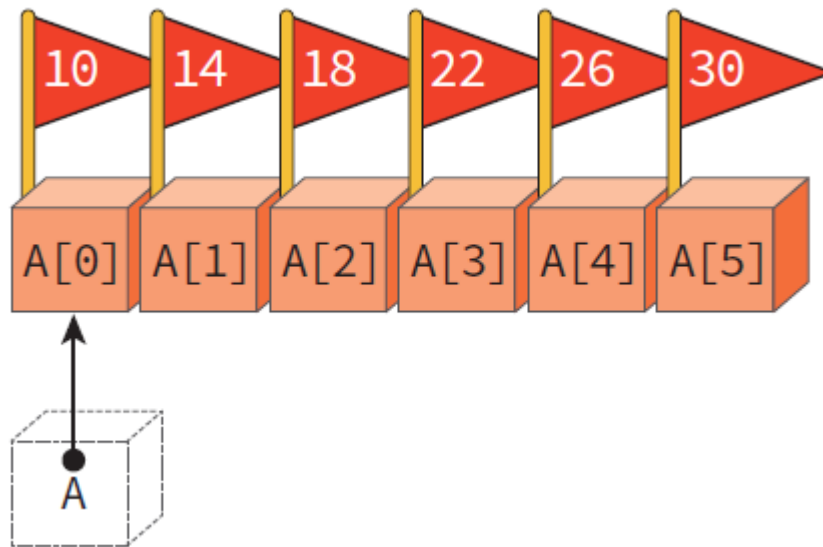
```
}
```





배열과 포인터

- 배열의 이름: 사실상의 포인터와 같은 역할





```
1  #include<stdio.h>
2  #define SIZE 6
3
4  void get_integer(int list[])
5  {
6      printf("6개의 정수를 입력하시오:");
7      for (int i = 0; i < SIZE; i++)
8          scanf_s("%d", &list[i]);
9  }
10
11  int cal_sum(int list[])
12  {
13      int sum = 0;
14      for (int i = 0; i < SIZE; i++)
15          sum = sum + *(list + i);
16
17      return sum;
18  }
19
20  int main(void)
21  {
22      int list[SIZE];
23      get_integer(list);
24      printf("합 = %d\n", cal_sum(list));
25      return 0;
26  }
```



동적 메모리 할당

□ 동적(Dynamic) 메모리 할당

- ▣ 프로그램의 실행 도중에 메모리를 할당 받는 것
- ▣ 필요한 만큼만 할당을 받고 또 필요한 때에 사용하고 반납
- ▣ 메모리를 매우 효율적으로 사용가능
- ▣ 임베디드 시스템에서 메모리 매핑시 사용

□ 정적(Static)메모리

- ▣ 컴파일 단계에서 메모리 할당 받는 것



동적 메모리 할당 : malloc()

□ 전형적인 동적 메모리 할당 코드

```
main()
{
    int *pi;
    pi = (int *)malloc(sizeof(int));    // 동적 메모리 할당
    ...
    ...                                // 동적 메모리 사용
    ...
    free(pi);                          // 동적 메모리 반납
}
```



동적 메모리 할당 예제

// MALLOC.C: malloc을 이용하여 정수 10를 저장할 수 있는 동적 메모리를
// 할당하고 free를 이용하여 메모리를 반납한다.

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <malloc.h>
```

```
#define SIZE 10
```

```
int main(void)
```

```
{
```

```
    int *p;
```

```
    p = (int *)malloc(SIZE * sizeof(int));
```

```
    if (p == NULL) {
```

```
        fprintf(stderr, "메모리가 부족해서 할당할 수 없습니다.\n");
```

```
        exit(1);
```

```
    }
```

```
    for (int i = 0; i < SIZE; i++)
```

```
        p[i] = i;
```

```
    for (int i = 0; i < SIZE; i++)
```

```
        printf("%d ", p[i]);
```

```
    free(p);
```

```
    return 0;
```

```
}
```