# Chapter 8. Pointers

May, 2016

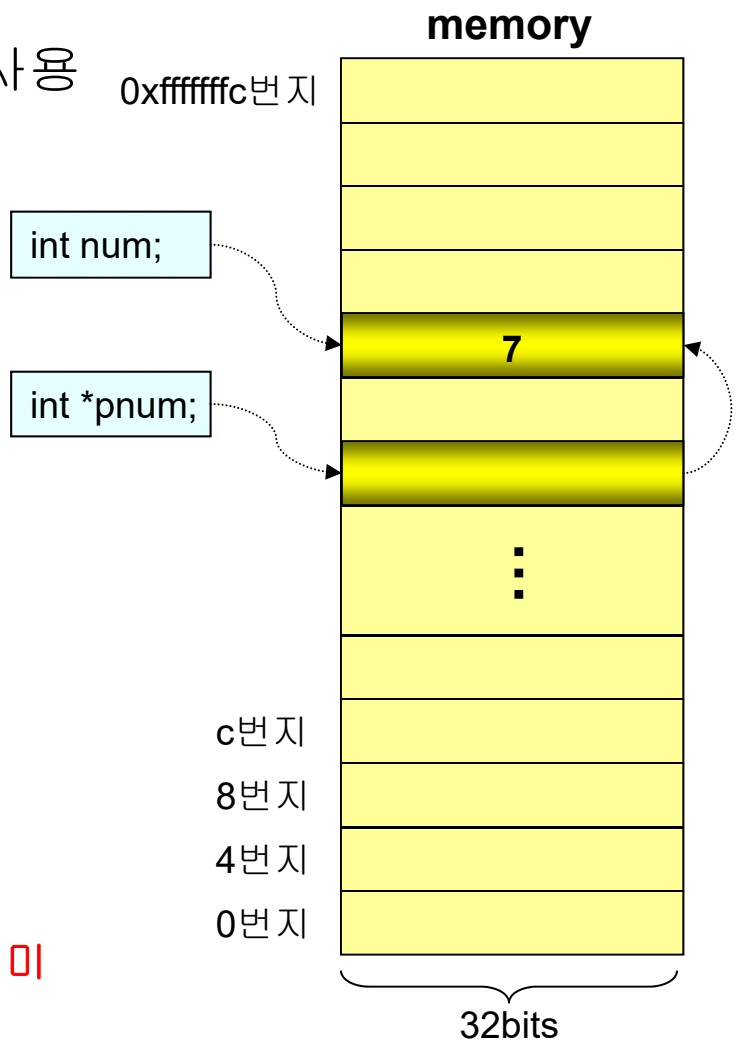
Seungjae Baek

Dept. of software
Dankook University
http://embedded.dankook.ac.kr/~baeksj

# 포인터의 개념 (1/6)

- 포인터란: 다른 객체를 가리키는 변수
  - ✓ 객체의 메모리 주소를 저장하는 변수
  - ✓ 기호적 방식(symbolic way)으로 주소 사용
- 포인터와 관련된 연산자
  - ✓ &
  - ✓ *

```
#include <stdio.h>
int main()
{
        int num;
        int *pnum;

        pnum = &num;
        *pnum = 7;
        printf("%d\n", num);
}
```

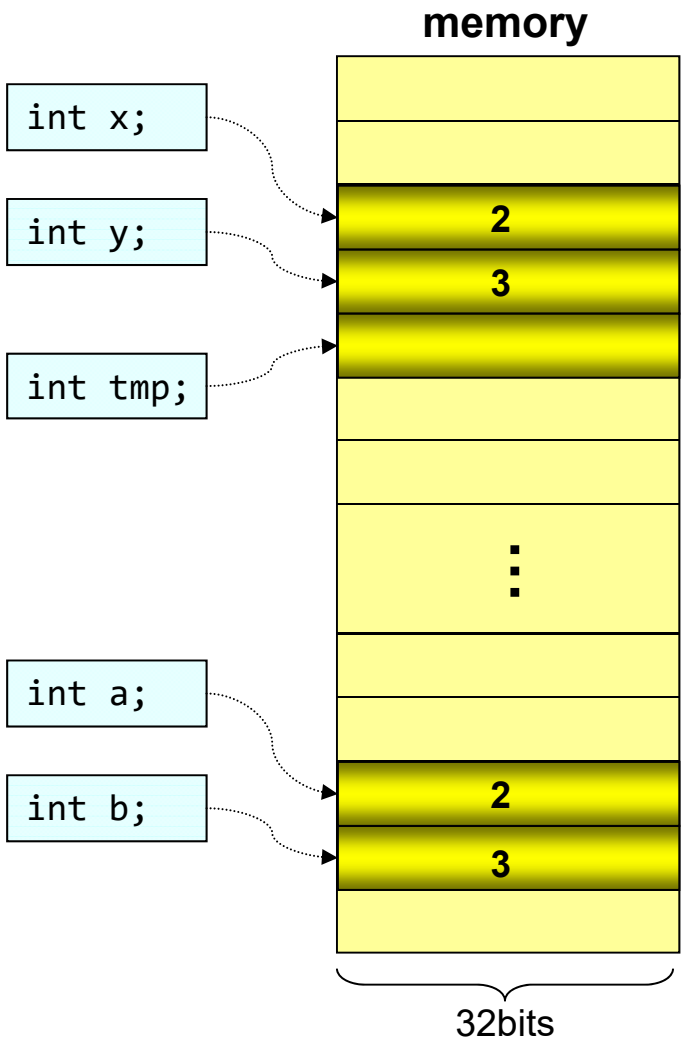



☞ **p**는 포인터 자신, ***p**는 포인터가 가리키는 객체를 의미

# 포인터의 개념 (2/6)

- 포인터를 사용하는 이유

```
#include <stdio.h> // 매개 변수 예제 프로그램 1

void swap(int, int);

void swap(int x, int y)
{
        int tmp;

        tmp = x;
        x = y;
        y = tmp;
}

int main()
{
        int a=2, b=3;

        printf("a=%d, b=%d\n", a, b);
        swap(a,b);
        printf("a=%d, b=%d\n", a, b);
}
```

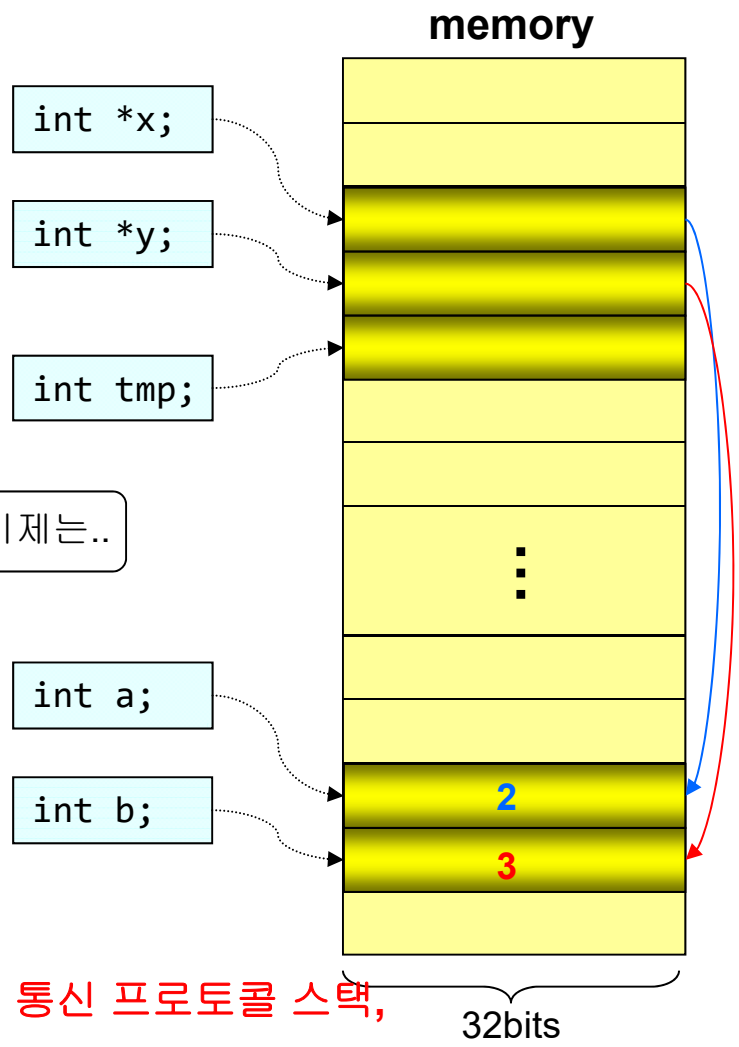


# 포인터의 개념 (3/6)

- 포인터를 사용하는 이유

```
#include <stdio.h>  // 매개 변수 예제 프로그램 2

void swap(int *, int *);

void swap(int *x, int *y)
{
        int tmp;

        tmp = *x;
        *x = *y;
        *y = tmp;
}

int main()
{
        int a=2, b=3;

        printf("a=%d, b=%d\n", a, b);
        swap(&a, &b);
        printf("a=%d, b=%d\n", a, b);
}
```

scanf()에서 &사용한 이유를 이제는..




☞ 포인터는 함수 인자, 문자열 처리, 리스트, 트리, 구조체, 통신 프로토콜 스택, 가상 함수, 해쉬 등에서 매우 중요하게 사용됨.

# 포인터의 개념 (4/6)

- 포인터를 사용 시 주의 사항
  - ✓ 객체를 가리킨 이후에 사용 가능

```
#include <stdio.h>
int main(void)
{
        int q = 100;
        int *p;

        *p = 100;
        printf("%d\n", *p);
        printf("%d\n", q);
        return 0;
}
```

```
[root@localhost C]# gcc -o pointer5 pointer5.c -Wall
pointer5.c: In function 'main':
pointer5.c:8:5: warning: 'p' is used uninitialized in this function [-Wuninitialized]
  *p = 100;
     ^
[root@localhost C]# gcc -o pointer5 pointer5.c -Wall -Werror
pointer5.c: In function 'main':
pointer5.c:8:5: error: 'p' is used uninitialized in this function [-Werror=uninitialized]
  *p = 100;
     ^
cc1: all warnings being treated as errors
[root@localhost C]#
```

# 포인터의 개념 (5/6)

- 포인터를 사용 시 주의 사항
  - ✓ 포인터가 가리키는 객체의 유형은 포인터 유형과 동일해야

```
#include <stdio.h>
int main(void)
{
        double q;
        char ch;
        int *p;

        ch = 'A';
        p = &ch;
        printf("%x %x\n", ch, *p);

        q = 1234.56;
        p = &q;
        printf("%lf, %lf\n", q, *p);
        return 0;
}
```

compile시 warning이 발생..

```
[root@localhost C]# gcc -o pointer6 pointer6.c
pointer6.c: In function 'main':
pointer6.c:9:4: warning: assignment from incompatible pointer type [enabled by d
efault]
  p = &ch;
    ^
pointer6.c:13:4: warning: assignment from incompatible pointer type [enabled by
default]
  p = &q;
    ^
[root@localhost C]# gcc -o pointer6 pointer6.c -Wall -Werror
pointer6.c: In function 'main':
pointer6.c:9:4: error: assignment from incompatible pointer type [-Werror]
  p = &ch;
    ^
pointer6.c:13:4: error: assignment from incompatible pointer type [-Werror]
  p = &q;
    ^
pointer6.c:14:2: error: format '%lf' expects argument of type 'double', but argu
ment 3 has type 'int' [-Werror=format=]
  printf("%lf, %lf\n", q, *p);
  ^
cc1: all warnings being treated as errors
[root@localhost C]#
```

# 포인터의 개념 (6/6)

- 포인터가 가리키는 공간의 동적 할당/해제

```
#include <stdio.h>
#include <stdlib.h>  // 동적 할당과 해제를 위해 추가

main()
{
        int *pi;

        pi = malloc(4);
        *pi = 1234;
        printf("%d\n", *pi);
        free(pi);
}
```

반드시 free 필요.

# 포인터 식 (1/4)

- 포인터 관련 연산자
  - ✓ *, &, +, ++, -, --
  - ✓ 포인터의 각 원소는 포인터 식으로 접근

```
#include <stdio.h>
int main(void)
{
        char *pch, arr_ch[] = "ABCDEFG";
        int *pi, arr_int[] = {10, 20, 30, 40, 50};
        double *pd, arr_double[] = {1.1, 2.2, 3.3, 4.4, 5.5};

        pch = arr_ch;
        pi = arr_int;
        pd = arr_double;

        printf("%c, %c\n", *pch, arr_ch[0]);
        printf("%d, %d\n", *pi, arr_int[0]);
        printf("%lf, %lf\n\n", *pd, arr_double[0]);

        printf("%c, %c\n", *(pch+2), arr_ch[2]);
        printf("%d, %d\n", *(pi+2), arr_int[2]);
        printf("%lf, %lf\n\n", *(pd+2), arr_double[2]);

        pch = (char *)200; printf("%d\n", pch+2);
        pi = (int *)200; printf("%d\n", pi+2);
        pd = (double *)200; printf("%d\n", pd+2);
        return 0;
}
```

배열 이름은 실제로 배열의 시작 주소이다!!.
따라서 포인터에 치환할 때 주소 연산자 없음

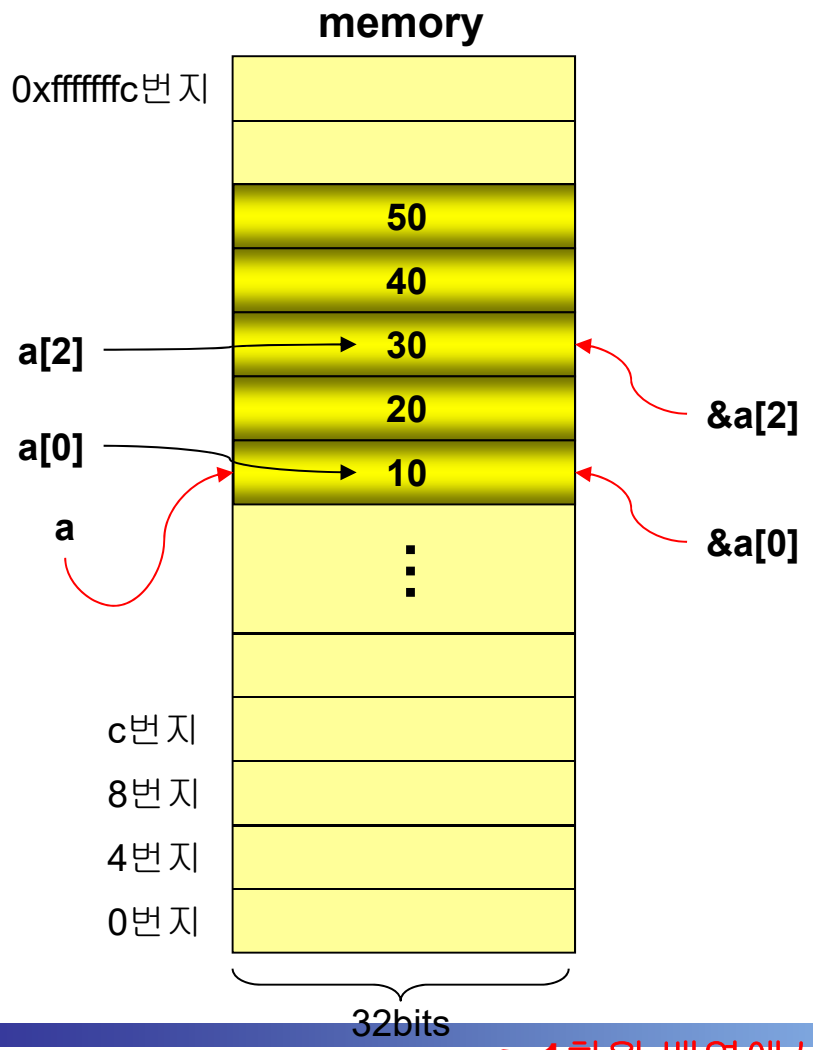
포인터 식의 모양에 주의

결국 포인터 수식은 포인터의
유형에 따라 다른 결과를 갖는다
(프로그래머의 편의성을 위해)

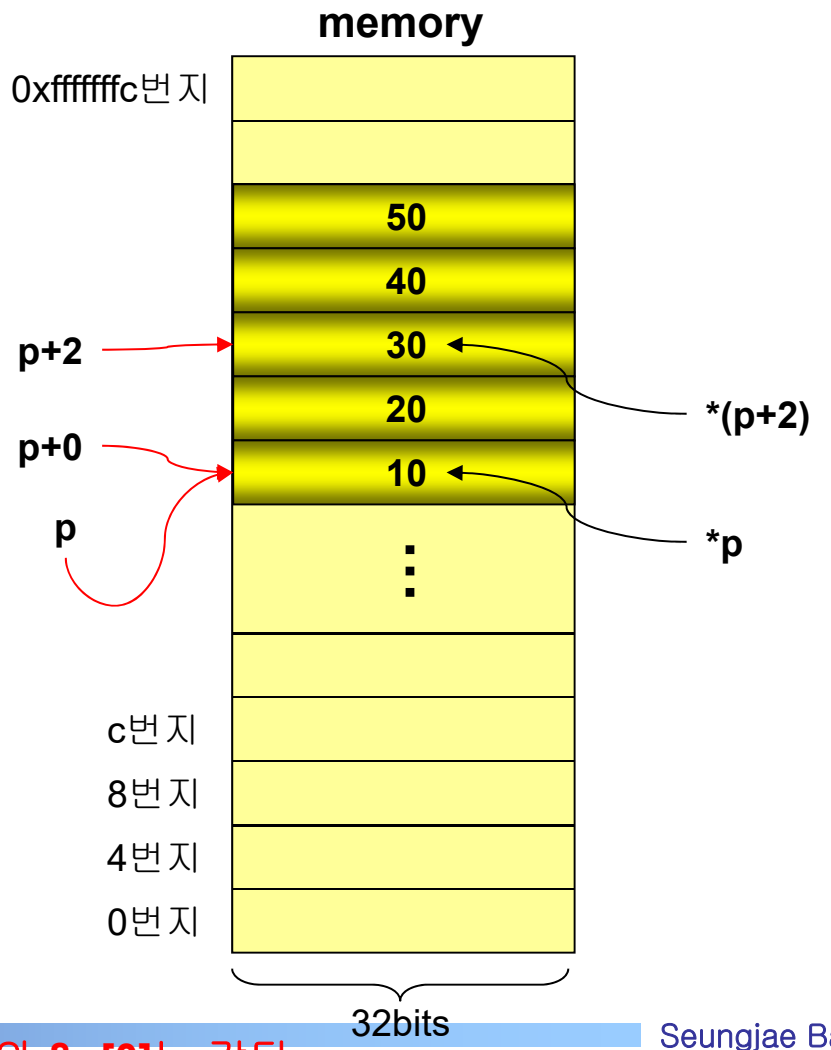
# 포인터 식 (2/4)

- 포인터 식

```
int a[] = {10, 20, 30, 40, 50};
```




```
int *p;
p = a;
```




☞ **1차원 배열에서 a와 &a[0]는 같다.**

# 포인터 식 (3/4)

- 포인터 식에 익숙해 지자!!

```
#include <stdio.h>
int main(void)
{
        int arr_int[] = {10, 20, 30, 40, 50};
        int *pi;

        pi = arr_int;
        return 0;
}
```

✓ 20을 포인터 식으로 출력하려면?

```
printf("%d\n", *(pi+1));
```

✓ 40을 포인터 식으로 출력하려면?

```
printf("%d\n", *(pi+3));
```

```
printf("%d\n", *pi+3);를 하면?
```

```
printf("%d\n", *(pi+5));를 하면?
```

포인터의 강력한 동시에
개발자가 많이 장점이며 실수하는 부분

# 포인터 식 (4/4)

- 포인터 연산자의 우선 순위

```
#include <stdio.h>

int main(void)
{
        int *pi, arr_int[] = {10, 20, 30, 40, 50};
        int result;

        pi = arr_int;

        result = *pi++;

        printf("%d, %d, %d\n", *pi, arr_int[0], arr_int[1]);
        printf("result = %d\n", result);
        printf("%d\n", *(pi+1));
        return 0;
}
```

result = (*pi)++;를 하면?

☞ 포인터 수식은 곱셈, 나눗셈은 허용 안됨. 실수 덧셈도 허용 안됨

# 배열과 포인터 (1/4)

- 배열은 포인터와 유사한 성격을 갖는다
  - ✓ 색인 없이 배열 이름을 사용하면, 배열 이름은 그 배열에 대한 포인터 기능을 수행 (배열의 이름은 포인터 상수)
    - 배열 이름을 그대로 (주소 연산자 없이) 포인터에 치환 가능
    - 색인 없는 배열 이름에 포인터 연산 적용 가능
    - 포인터가 배열을 가리킬 경우, 포인터에 색인 적용 가능

```
#include <stdio.h>
int main(void)
{
        int a[5] = {10, 20, 30, 40, 50};
        int *p;

        printf("%d %d %d\n", a[0], a[2], a[4]);

        p = a;

        printf("%d %d %d\n", *p, *(p+2), *(p+4));

        printf("%d %d %d\n", *a, *(a+2), *(a+4));

        printf("%d %d %d\n", p[0], p[2], p[4]);
}
```

p = &a[0];도 가능

printf("%d %d %d\n", *p, *p+2, *p+4);

# 배열과 포인터 (2/4)

- 배열과 포인터에서 주의 사항
  - ✓ 포인터는 가리키는 위치를 수정할 수 있지만, 배열은 안됨
  - ✓ 포인터 명 자체는 메모리를 차지, 배열 명 자체는 메모리를 차지하지 않음

```
#include <stdio.h>  // 배열과 포인터의 차이점
int main(void)
{
        int a[5] = {10, 20, 30, 40, 50};
        int *p;

        printf("%d\n", a[0]);
        printf("%d\n", a[1]);

        p = a;

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

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

배열의 이름은 포인터 상수
**error: lvalue required as increment operand**

☞ 포인터가 일반적이고 강력함. 배열은 직관적이라 배우기 쉽고 오류 가능성이 적음

# 배열과 포인터 (3/4)

- 2차원 배열을 포인터로 접근

```
#include <stdio.h>   // 2차원 배열을 포인터로 접근

int main(void)
{
        int a[5][2] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
        int *p;

        p = a;

        printf("%d\n", *p);
        printf("%d\n", *(p+1));
        printf("%d\n", *p+1);

        printf("%d\n", *(p+2*2+1));
        printf("%d\n", *(p+4*2+0));

        p = a[2];

        printf("%d\n", *p);
        printf("%d\n", *(p+1));
        return 0;
}
```

p = &a[0][0];
p = a[0];

*(시작 주소+첫번째 색인*열의 전체 개수+ 두번째 색인)

# 배열과 포인터 (4/4)

- 포인터 사용 예
  - ✓ compiler에서 lexical analysis (어휘 분석)
    - 입력 문장을 각 token으로 구분
    - 예를 들어 “Pointers are fun to use”문장을 입력 받아 token으로 구분

```
#include <stdio.h>  // lexical analysis
#include <ctype.h>

int main(void)
{
        char input[80]; char tmp_buf[12]; char *p, *q;

        printf("Enter any Sentences\n");     gets(input);

        p = input; q = tmp_buf;
        while (*p) {
                if (isalnum(*p))
                        *q++ = *p++;
                else {
                        *q = '\0'; printf("token = %s\n", tmp_buf);
                        p++; q = tmp_buf;
                }
        }
        *q = '\0'; printf("token = %s\n", tmp_buf);
        return 0;
}
```

이러한 프로그램 방식에 익숙해 져야..

# 문자열 상수 포인터

- 문자열 상수는 컴파일러가 별도 테이블(메모리)에서 관리

```
#include <stdio.h> // 문자열 상수 포인터 예제

int main(void)
{
        char *pch;
        pch = "Is it correct?";
        printf("%c\n", *pch);
        return 0;
}
```

```
char [] pch = "Is it correct?";
```

✓ 문자열 상수는 결국 문자열을 가리키는 주소

```
printf("나의 C 실력은 %c\n", *"Fantastic");
```

```
printf("나의 C 실력은 %s\n", (char *)"Fantastic");
```

☞ 이제는 **printf()**에서 **%c**와 **%s**의 차이를 말할 수 있다**!!**

# 포인터 배열

- 다른 기본 자료형처럼 포인터도 배열로 만들 수 있음

```
#include <stdio.h>
int main(void)
{
        int *i[3];
        int a=1, b=2, c=3;

        i[0] = &a;
        i[1] = &b;
        i[2] = &c;

        printf("%d, %d, %d\n", *i[0], *i[1], *i[2]);
        return 0;
}
```

☞ 예제의 변수 상태를 그림으로 그려보세요.

# 다중 간접 참조 (1/3)

- 포인터의 포인터

```
#include <stdio.h>  // 다중 참조의 간단한 예
int main(void)
{
        int i, *pi, **ppi;

        pi = &i;
        ppi = &pi;

        i = 10;
        printf("%d, %d\n", *pi, **ppi);
        return 0;
}
```

결국 pi, *ppi는 &i 값을 갖는다.

# 다중 간접 참조 (2/3)

- 포인터 배열과 다중 간접 참조의 예
  - ✓ 10명의 학생에 국어, 영어, 수학 성적 출력

```
#include <stdio.h>    // 포인터의 배열 예 2
int main(void)
{
    int kor[10]  = {70, 80, 90, 75, 85, 95, 78, 88, 98, 100};
    int eng[10]  = {78, 93, 88, 65, 70, 80, 90, 75, 85, 95};
    int math[10] = {75, 85, 78, 93, 88, 98, 60, 70, 80, 90};
    int *course[3];  // 포인터의 배열

    course[0] = kor;
    course[1] = eng;
    course[2] = math;

    printf("%d\n", *(course[1]+2));
    printf("%d\n", *(course[2]+4));

    printf("%d\n", *(*(course+1)+1));
    printf("%d\n", *(*(course+2)+3));

    printf("%d번의 세과목 성적: %d, %d, %d\n", 0,
        *(*(course+0)+0), *(*(course+1)+0), *(*(course+2)+0));
    printf("%d번의 세과목 성적: %d, %d, %d\n", 7,
        *(*(course+0)+7), *(*(course+1)+7), *(*(course+2)+7));
    return 0;
}
```

이 문제의 경우 대부분 프로그램에서는 2차원 배열 사용. (옆 프로그램은 강의를 위해 인위적으로..)

```
#include <stdio.h>            // 2차원 배열을 사용하면

main()
{
    int course[3][10] = {{70, 80, 90, 75, 85, 95, 78, 88, 98, 100},
                         {78, 93, 88, 65, 70, 80, 90, 75, 85, 95 },
                        {75, 85, 78, 93, 88, 98, 60, 70, 80, 90 }};

    printf("%d번의 세과목 성적: %d, %d, %d\n", 0,
              course[0][0], course[1][0], course[2][0]);
    printf("%d번의 세과목 성적: %d, %d, %d\n", 7,
             *(*(course+0)+7), *(*(course+1)+7), *(*(course+2)+7));
}
```

# 다중 간접 참조 (3/3)

- 포인터 배열과 다중 간접 참조의 예
  - ✓ 문자열 상수 배열이 포인터 배열로 실제 많이 사용됨

```
#include <stdio.h>    // 악의 축 찾기 프로그램
#include <ctype.h>
#include <string.h>

char *axis_of_evil[] = {"iraq", "iran",
"northkorea", ""};
char input[80], *p;

void next_token(char *q)
{
    while (*p) {
        if (isalnum(*p))
           *q++ = *p++;
        else {
           *q = '\0';
           if (*p != '\0')
               p++;
               return;
        }
    }
    *q = '\0';
    return;
}
```

```
int main(void)
{
    char buf[20];
    int i;

    printf("Enter any Sentences\n");
    gets(input);

    p = input;
    next_token(buf);
    while (strcmp(buf, "")) {
        for (i=0; strcmp(axis_of_evil[i], ""); i++)
           if (!strcmp(axis_of_evil[i], buf))
               printf("%s is found in \"%s\“ \n",
                      axis_of_evil[i], input);
        next_token(buf);
    }
    return 0;
}
```

# 포인터 매개 변수

- 함수의 인자로 포인터를 사용하는 경우
  - ✓ 호출된 함수에서 호출한 함수의 변수 값을 변화시키려면 포인터를 사용 ➔ 4 페이지 예제 참조,
  - ✓ scanf() 및 구조체의 경우
  - ✓ 배열

```
#include <stdio.h>        // 포인터 매개 변수 (배열)
void strcpy_new(char *p, char *q)
{
        while (*q)
                *p++ = *q++;
        *p = '\0';
}
int main(void)
{
        char dest[40], src[40] = “This is the last slide.";
        strcpy_new(dest, src);
        puts(dest);
        return 0;
}
```

# 이 장의 결론

- 포인터의 개념을 이해.
- 포인터 식 이해.
- 배열과 포인터의 공통점과 차이점 이해.
- 문자열 상수 포인터를 이해.
- 다중 간접 참조를 이해.
- 포인터 매개 변수를 이해.

# 참고: 포인터 이해에 도움이 될만한 예제

```
#include <stdio.h>        // 포인터의 포인터 (도움이 되기를….^^)

void func1(int x[][3])
{
        printf("%d\n", **x);
        printf("%d\n", **x+1);
        printf("%d\n", **(x+1));
        printf("%d\n", *(*x+1));
        printf("%d\n", *(*(x+1)+2));
}

main()
{
        int a[2][3] = {10, 20, 30, 40, 50, 60};

        func1(a);
}
```

# 참고: 함수 포인터

```
#include <stdio.h>        // 함수 포인터

int add(int x, int y)
{
        printf("%d + %d = %d\n", x, y, x+y);
        return 0;
}

main()
{
        int (*p)(int, int);

        p = add;

        p(2, 3);
}
```

# 참고: 배열 포인터

```
#include <stdio.h>       // 2차원 배열을 포인터 식으로 접근하는 방법

main()
{
        int a[5][2] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
        int *p;                  // 1차원 배열
        int *q[2];               // 포인터의 배열
        int (*r)[2];             // 배열의 포인터

        p = a;                   // 1. 포인터로 접근
        printf("%d\n", *p);
        printf("%d\n", *(p+1));
        printf("%d\n", *(p+2*2+1));

        printf("%d\n", **a);     // 2. 배열 이름에 포인터 식 적용
        printf("%d\n", *(*a+1));
        printf("%d\n", *(*(a+2)+1));

        q = a;                   // 3. 포인터의 배열 사용하면
        printf("%d\n", **q);
        printf("%d\n", *(*q+1));
        printf("%d\n", *(*(q+2)+1));

        r = a;                   // // 4. 배열의 포인터 사용하면
        printf("%d\n", **r);
        printf("%d\n", *(*r+1));
        printf("%d\n", *(*(r+2)+1));
}
```

컴파일 시 에러 발생.
당연!!. 배열에는 새로운 값을 치환할 수 없다.

# 참고: 이런 모양도 가능??

```
#include <stdio.h>  // 포인터 식에서 교환 법칙은? 배열에서는?

main()
{
        int a[5] = {10, 20,};

        printf("a[1] = %d\n", a[1]);
        printf("1[a] = %d\n", 1[a]);
        printf("*(a+1) = %d\n", *(a+1));
        printf("*(1+a) = %d\n", *(1+a));
}
```