C 언어 스터디 6주차

복습

```
13  void BinarySearch(int arr[], int first, int last, int key)
14  {
15      int middle = (first + last) / 2;
16      if (first > last)
17      {
18          printf("찾는 값이 없습니다.\n");
19      }
20      else if (arr[middle] == key)
21      {
22          printf("탐색 성공, index : %d\n", middle);
23      }
24      else if (arr[middle] < key)
25      {
26          BinarySearch(arr, middle + 1, last, key);
27      }
28      else
29      {
30          BinarySearch(arr, first, middle - 1, key);
31      }
32  }
```

복습

```c
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
void make_maze(int maze[][12]);
int decide_direct(int A[][12], int now_i, int now_j);
void print_maze(int A[][12]);
#define BLOCK 2
#define WALK 1
int count = 0;

int main()
{
    int direct, win = 0;
    int maze[12][12] = { 0 };           //미로 배열
    int now_i = 1, now_j = 1;

    srand((unsigned)time(NULL));
    make_maze(maze);
```

복습

```
20      while (count < 1000)
21      {
22          direct = decide_direct(maze, now_i, now_j)
23          if (direct == 0)
24          {
25              now_i--, now_j--;
26              maze[now_i][now_j] = WALK;
27          }
28          else if (direct == 1)
29          {
30              now_i--;
31              maze[now_i][now_j] = WALK;
32          }
33          else if (direct == 2)
34          {
35              now_i--, now_j++;
36              maze[now_i][now_j] = WALK;
37          }
```

```
38          else if (direct == 3)
39          {
40              now_j--;
41              maze[now_i][now_j] = WALK;
42          }
43          else if (direct == 4)
44          {
45              now_j++;
46              maze[now_i][now_j] = WALK;
47          }
48          else if (direct == 5)
49          {
50              now_i++, now_j--;
51              maze[now_i][now_j] = WALK;
52          }
53          else if (direct == 6)
54          {
55              now_i++;
56              maze[now_i][now_j] = WALK;
57          }
```

```
58          else
59          {
60              now_i++, now_j++;
61              maze[now_i][now_j] = WALK;
62          }
63
64          if ((now_i == 10) && (now_j == 10))
65          {
66              win = 1;
67              break;
68          }
69      }
```

```
71      print_maze(maze);
72      printf("\n");
73      if (win == 1)
74      {
75          printf("탈출했습니다. 시도 횟수 : %d\n", count);
76      }
77      else
78      {
79          printf("탈출에 실패했습니다.\n");
80      }
81
82      return 0;
```

복습

```
85  void make_maze(int maze[][12])
86  {
87      int i;
88      int tmp_i, tmp_j;
89
90      for (i = 0;i < 12;i++)
91      {
92          maze[i][0] = BLOCK;
93          maze[0][i] = BLOCK;
94          maze[i][11] = BLOCK;
95          maze[11][i] = BLOCK;
96      }
97      maze[1][1] = WALK;
```

```
99      for (i = 0;i < 20;)
100     {
101         tmp_i = rand() % 10 + 1;
102         tmp_j = rand() % 10 + 1;
103         if ((maze[tmp_i][tmp_j] != BLOCK) && !(tmp_i == 1 && tmp_j == 1) && !(tmp_i == 10 && tmp_j == 10))
104         {
105             maze[tmp_i][tmp_j] = BLOCK;
106             i++;
107         }
108     }
109 }
```

```
111 int decide_direct(int A[][12], int now_i, int now_j)
112 {
113     int rand_num, result;
114     while (1)
115     {
116         count++;
117         rand_num = rand() % 100;
118 
119         if (rand_num < 2)           //왼쪽 위
120         {
121             result = 0;
122             if (A[now_i - 1][now_j - 1] != BLOCK)
123                 break;
124         }
125         else if (rand_num < 6)          //위
126         {
127             result = 1;
128             if (A[now_i - 1][now_j] != BLOCK)
129                 break;
130         }
131         else if (rand_num < 16)         //오른쪽 위
132         {
133             result = 2;
134             if (A[now_i - 1][now_j + 1] != BLOCK)
135                 break;
136         }
137         else if (rand_num < 20)         //왼쪽
138         {
139             result = 3;
140             if (A[now_i][now_j - 1] != BLOCK)
141                 break;
142         }
143         else if (rand_num < 40)         //오른쪽
144         {
145             result = 4;
146             if (A[now_i][now_j + 1] != BLOCK)
147                 break;
148         }
149         else if (rand_num < 50)         //왼쪽 아래
150         {
151             result = 5;
155         else if (rand_num < 70)         //아래
156         {
157             result = 6;
158             if (A[now_i + 1][now_j] != BLOCK)
159                 break;
160         }
161         else
162         {
163             result = 7;         //오른쪽 아래
164             if (A[now_i + 1][now_j + 1] != BLOCK)
165                 break;
166         }
167     }
168 
169     return result;
170 }
```

복습

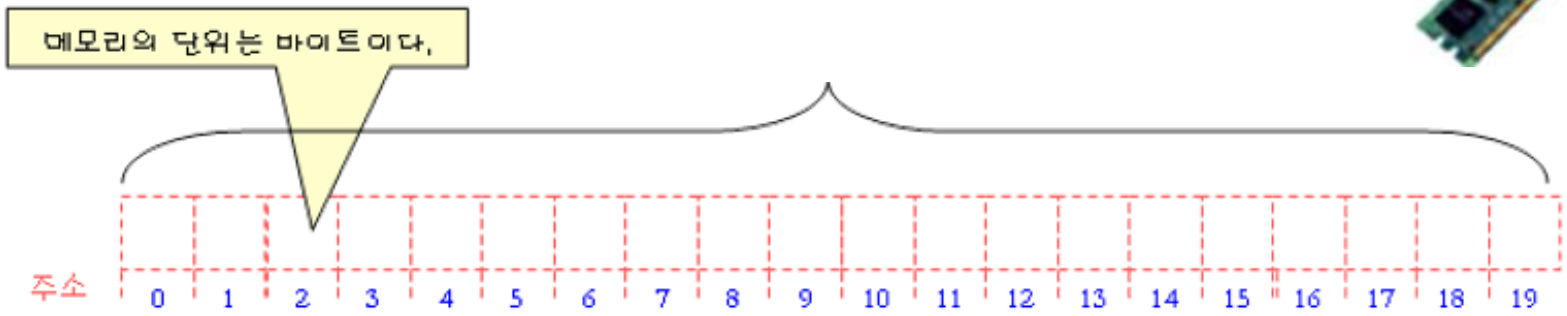
```
172  void print_maze(int A[][12])
173  {
174      int i, j;
175      for (i = 0;i < 10;i++)                    //테두리를 제외하고 출력
176      {
177          for (j = 0;j < 10;j++)
178          {
179              if (A[i + 1][j + 1] == 0)          //아무것도 아닌 경우
180              {
181                  printf(" . ");
182              }
183              else if (A[i + 1][j + 1] == WALK)   //이동 경로
184              {
185                  printf(" * ");
186              }
187              else
188              {
189                  printf(" & ");          //벽
190              }
191          }
192          printf("\n");
193      }
194  }
```

포인터

포인터란 메모리에 있는 **변수의 주소**를 가지고 있는 **변수**이다.

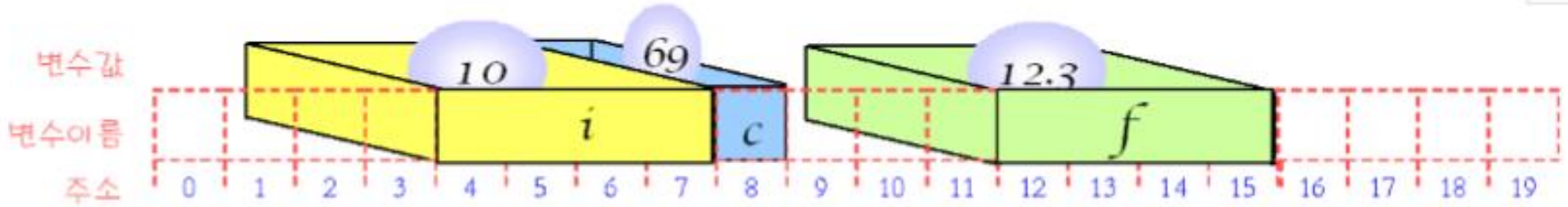
현실에서 모든 가정집이 주소를 갖고 있듯이 변수들 또한 주소를 갖는다.
이를 저장하는 변수가 바로 포인터이다.

변수는 메모리에 저장되고 바이트 단위로 액세스된다.

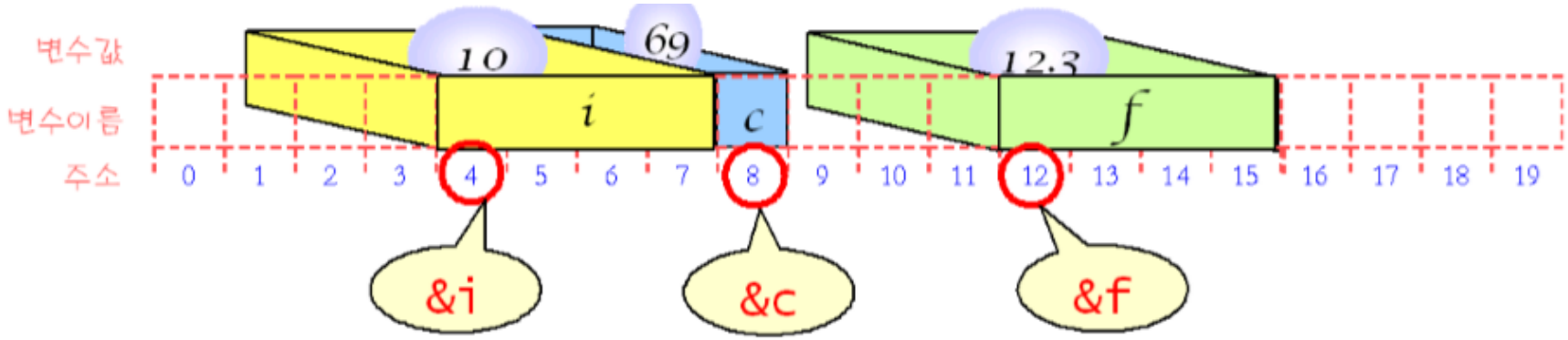


포인터

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

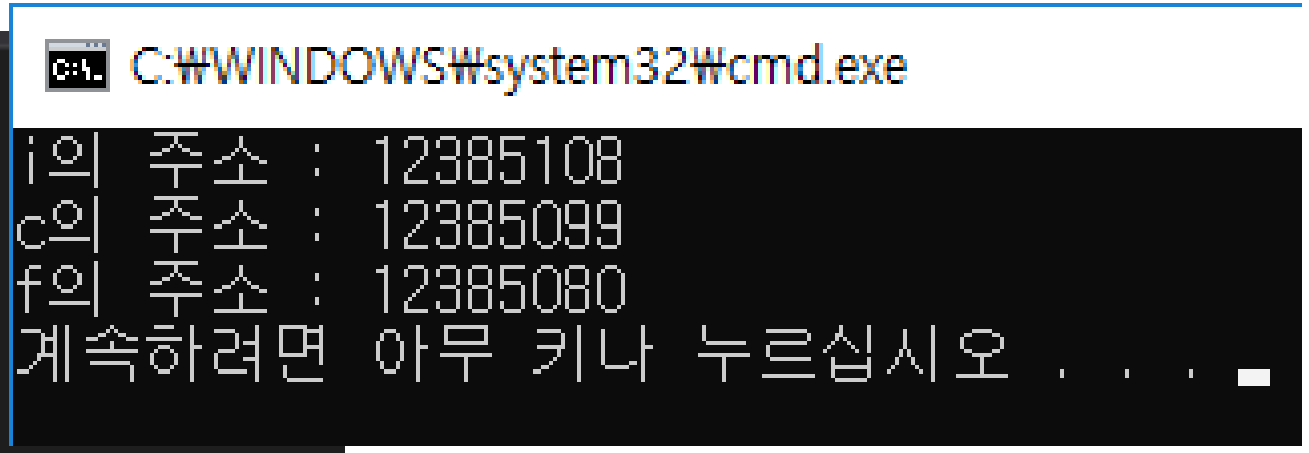


- 변수의 주소를 계산하는 연산자 : &
- 변수 i의 주소 : &i



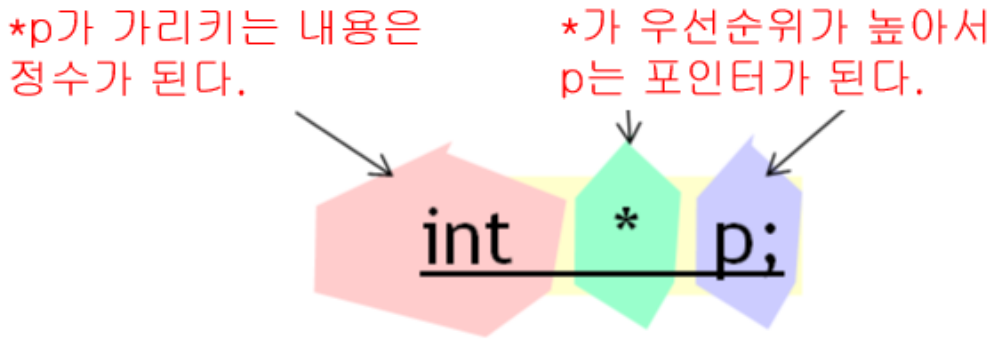
포인터

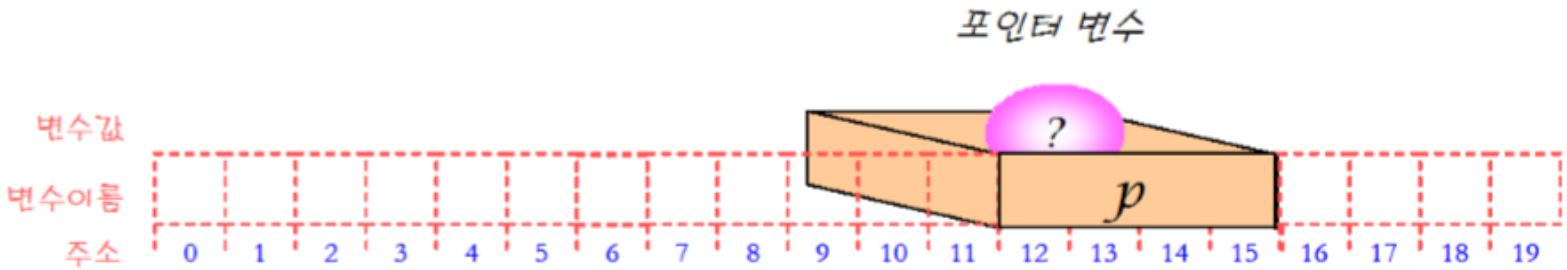
```c
#include<stdio.h>


int main() {
    int i = 10;
    char c = 60;
    double f = 12.3;
    printf("i의 주소 : %u\n", &i);
    printf("c의 주소 : %u\n", &c);
    printf("f의 주소 : %u\n", &f);
    return 0;
}
```



포인터 선언

- 포인터도 변수의 일종이다. 따라서 사용하기 전에 선언해야 한다.
- 일반적인 변수 선언처럼 선언하되, 변수이름 앞에 *를 붙인다.
- Ex) int *p; //int형 변수를 가리키는 포인터변수 p
- 여기서 *는 곱셈과 전혀 관련이 없다.





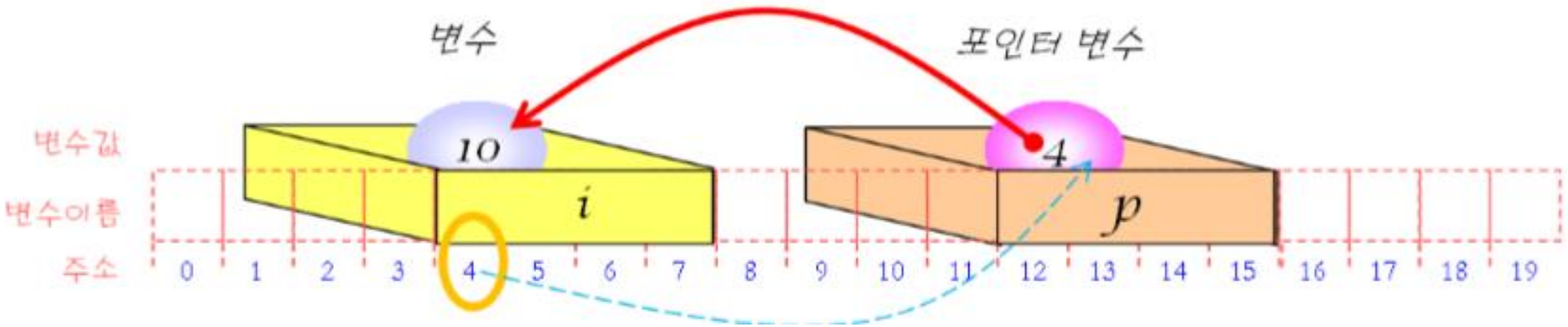
포인터와 변수 연결

- 포인터를 사용하기 전에는 초기화를 해주어야 한다.
- 포인터에는 변수의 주소가 저장되야 하므로 &를 이용한다.

Ex) int i = 10;

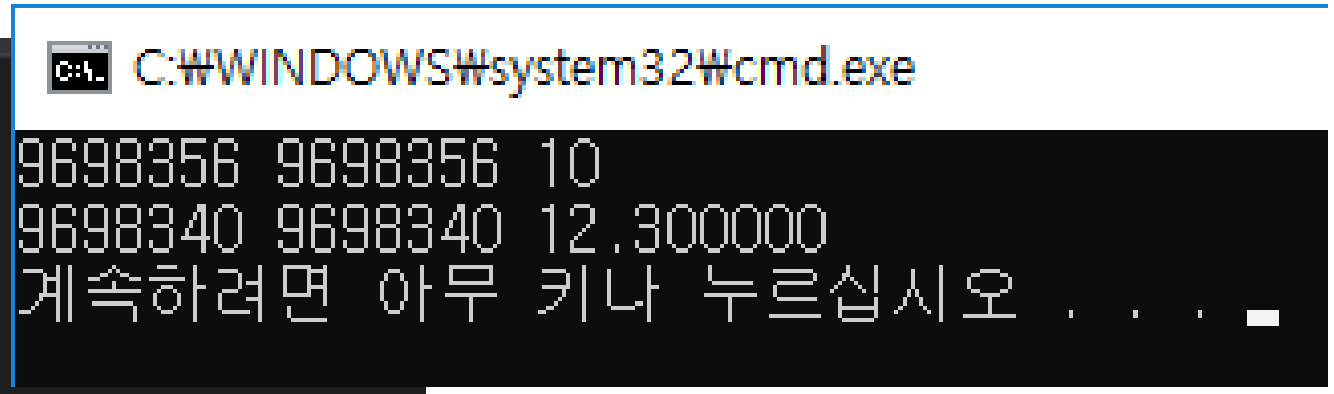
int *p = &i;

(아래에선 변수 i의 주소를 4라고 가정하였다.)



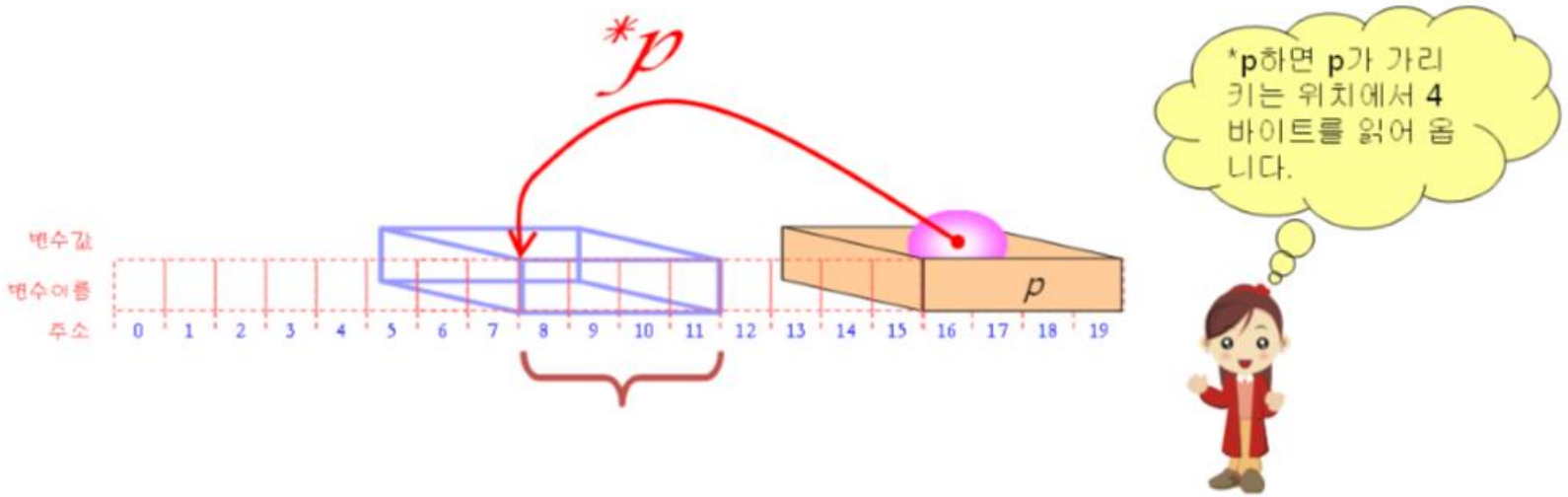
포인터와 변수 연결

```c
#include<stdio.h>


int main() {
    int i = 10;
    double f = 12.3;
    int *pi = NULL;
    double *pf = NULL;

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



간접 참조 연산자 *

- 포인터 p가 가리키는 주소에 저장된 내용을 가져오려면 *p처럼 사용한다.
- 이를 포인터를 통하여 **간접 참조** 한다고 한다.
- p가 i를 가리킨다면 *p == i
- 간접 참조 연산자를 사용하면 포인터가 가리키는 곳에 가서 자료형의 **크기만큼** 값을 읽어온다. //포인터에 자료형이 필요한 이유



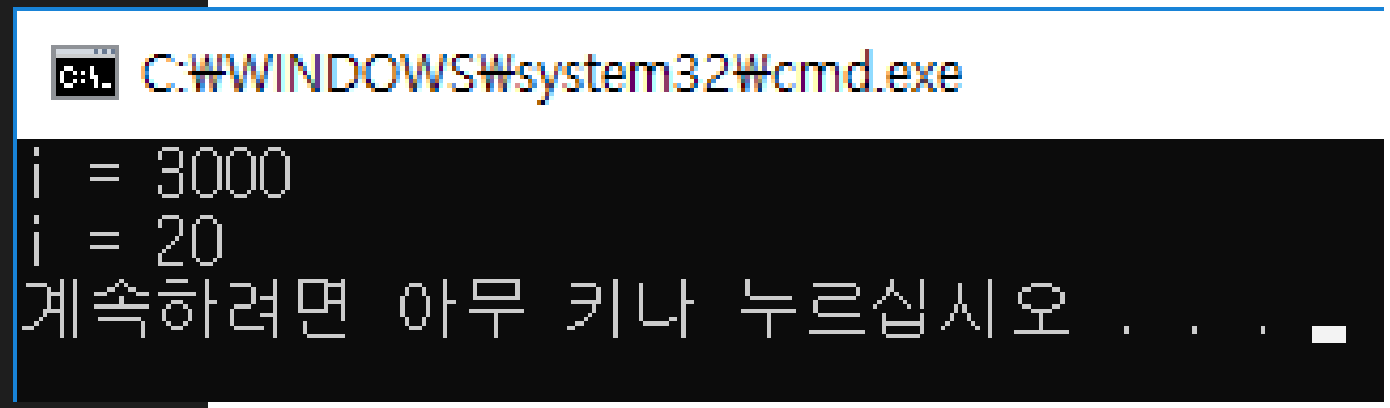
간접 참조 연산자 *

```c
#include<stdio.h>


int main() {
    int i = 3000;
    int *p = NULL;

    p = &i;

    printf("i = %d, &i = %u\n", i, &i);
    printf("*p = %d, p = %u\n", *p, p);
    return 0;
}
```

```
C:₩WINDOWS₩system32₩cmd.exe
i = 3000, &i = 12515392
*p = 3000, p = 12515392
계속하려면 아무 키나 누르십시오 . . .
```

간접 참조 연산자 *

```c
#include<stdio.h>


int main() {
    int i = 3000;
    int *p = NULL;

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

    *p = 20;
    printf("i = %d\n", i);
    return 0;
}
```



포인터 사용시 주의점

1. 포인터를 초기화 하지 않고 사용하는 것.

int *p; *p = 100; 와 같이 사용하면 X. 메모리 관리 권한은 운영체제가 가지고 있으므로 사용자는 p가 어디를 가리키는지 모름. 우연히 p가 중요한 부분을 가리키고 있었다면 시스템 다운까지도 유발 가능.

아무것도 가리키지 않을 때는 int *p = NULL; 처럼 초기화 해주자.

2. 포인터 타입과 변수의 타입은 일치시켜야 한다.

서로 다른 크기의 포인터와 변수를 사용한다면 간접 참조 시에 옆 메모리를 침범할 수 있다. 따라서 특별한 경우가 아니면 포인터로 다른 타입의 데이터를 가리키게 하면 안된다.

3. 절대 주소 사용 금지

포인터를 초기화 한다고 int *p = 10000;과 같이 값을 지정해주는 것은 바람직하지 않다.

값을 직접 대입하는 것은 특별한 경우에만 사용하자. 왜 안되는지는 1번과 같다.

포인터 연산

포인터 연산은 일반적인 변수의 연산과는 조금 다르다.

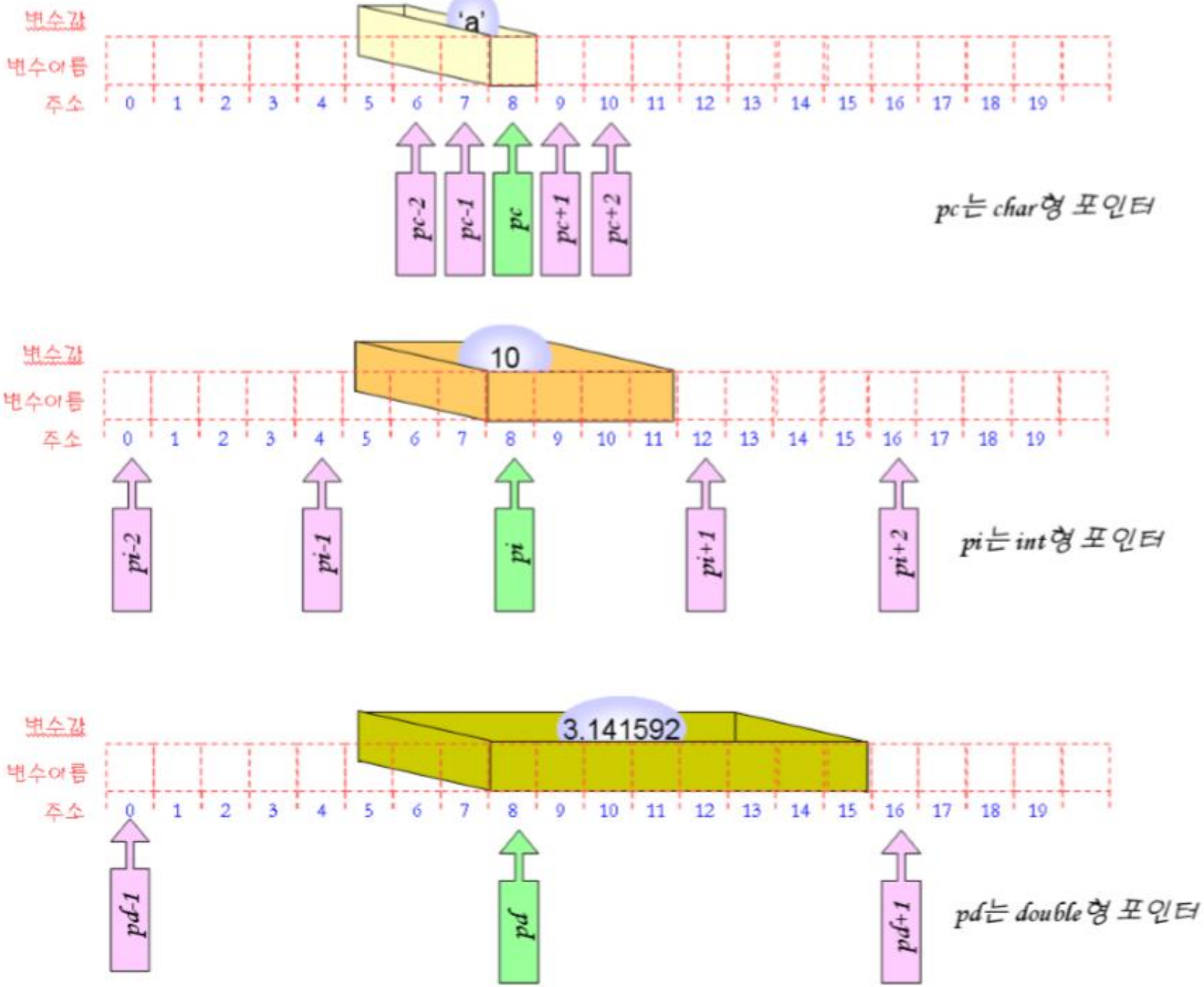
포인터에 증감연산자를 쓰거나 덧셈 뺄셈 연산을 한다면 증가되는 값은 포인터가 가리키는 객체의 크기이다.

즉 포인터가 가리키는 자료형의 크기가 s라면 s*n만큼 크기가 증가한다.

포인터 타입	++연산 후 증가되는 값
char	1
short	2
int	4
float	4
double	8

포인터 연

```
1   #includ
2
3
4   int mai
5       cha
6       int
7       dou
8       pc
9       pi
10      pd
11      //
12
13      pri
14
15      pc+
16      pri
17
18      pri
19      ret
20  }
```



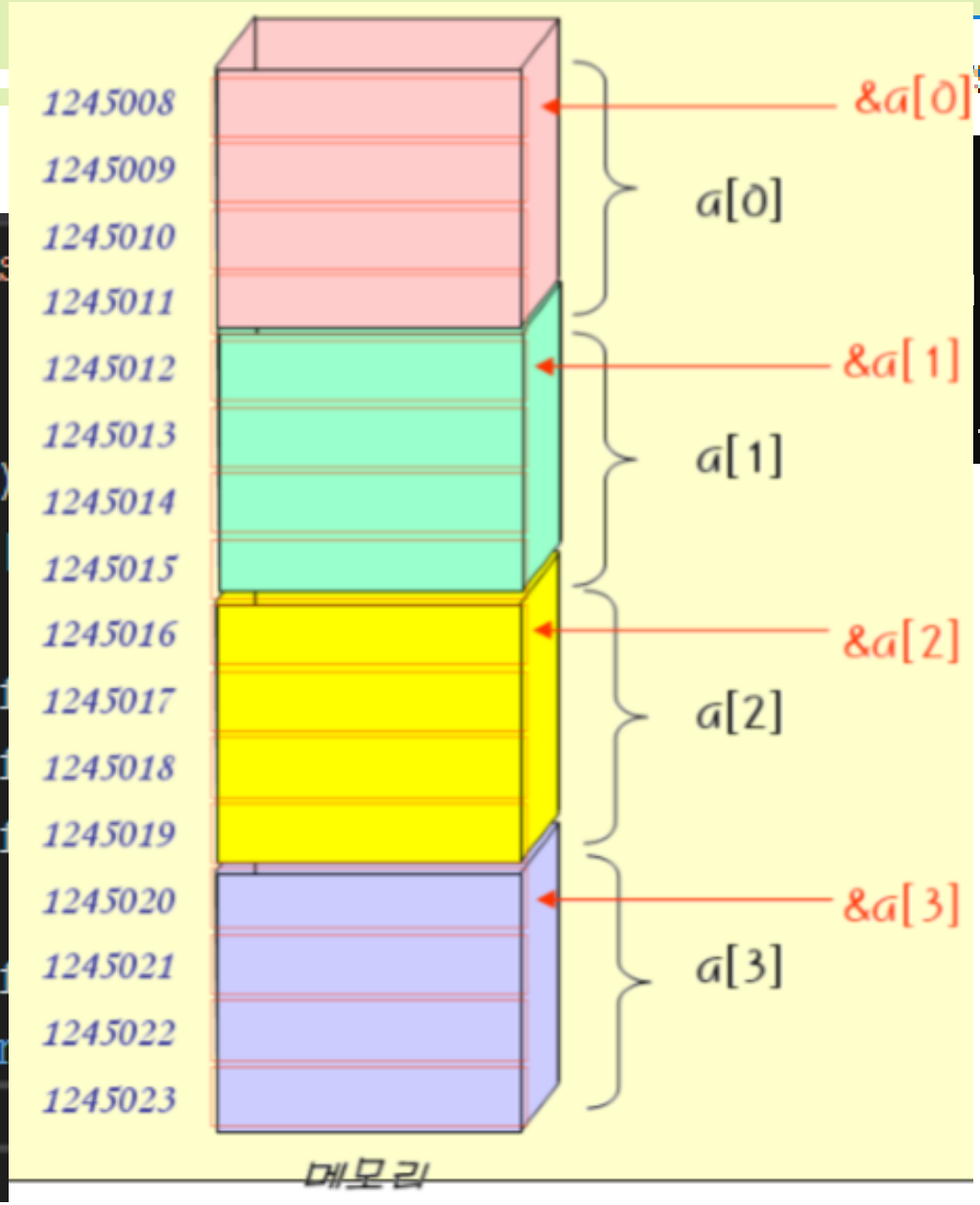

```
pd = 10000
pd = 10008
2 = 10024
. . .
```

간접 참조 연산자와 증감 연산자

수식	++연산 후 증가되는 값
v = *p++	p가 가리키는 값을 v에 대입한 후에 p를 증가한다.
v = (*p)++	p가 가리키는 값을 v에 대입한 후에 가리키는 값을 증가한다.
v = *++p	p를 증가시킨 후에 p가 가리키는 값을 v에 대입한다.
v = ++*p	p가 가리키는 값을 가져온 후에 그 값을 증가하여 v에 대입한다.

포인터와 배열

```
1   #include<s
2
3
4   int main()
5       int a[
6
7       printf
8       printf
9       printf
10
11      printf
12      return
13  }
```

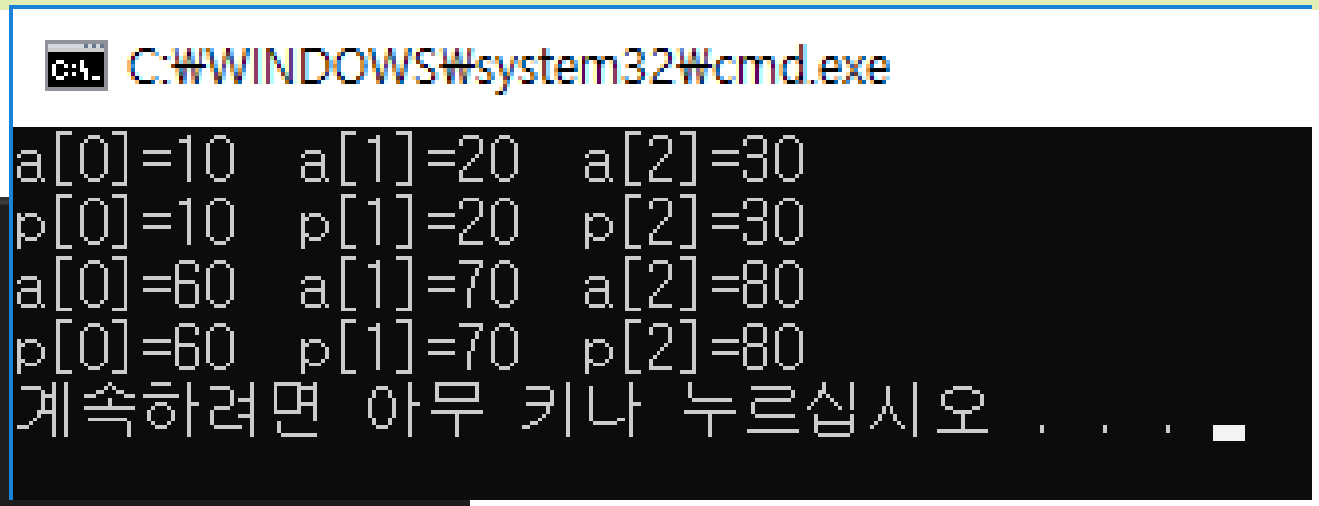


stem32₩cmd.exe

키나 누르십시오 . . .

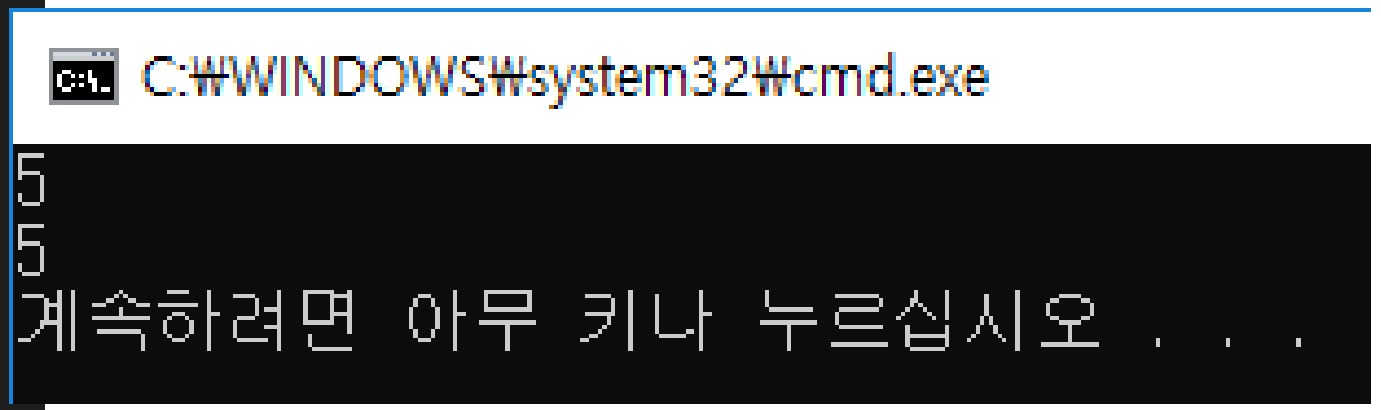
포인터와 배열

```c
#include<stdio.h>


int main() {
    int a[] = { 10,20,30,40,50 };
    int *p;
    p = a;
    printf("a[0]=%d  a[1]=%d  a[2]=%d\n", a[0], a[1], a[2]);
    printf("p[0]=%d  p[1]=%d  p[2]=%d\n", p[0], p[1], p[2]);

    p[0] = 60;
    p[1] = 70;
    p[2] = 80;

    printf("a[0]=%d  a[1]=%d  a[2]=%d\n", a[0], a[1], a[2]);
    printf("p[0]=%d  p[1]=%d  p[2]=%d\n", p[0], p[1], p[2]);
    return 0;
}
```



포인터와 함수

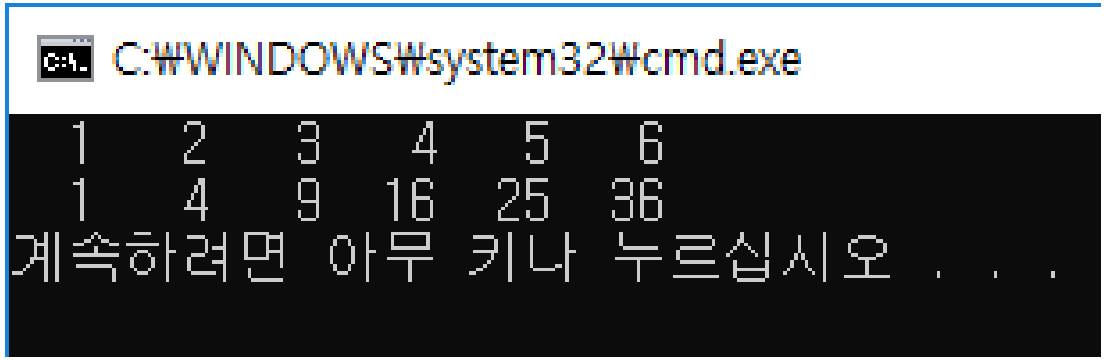
```c
#include<stdio.h>

void square(int a) {
    a = a * a;
}

int main() {
    int x = 5;

    printf("%d\n", x);
    square(x);
    printf("%d\n", x);
    return 0;
}
```



일반 변수는 값에 의한 호출로,
원본은 변하지 않는다.

포인터와 함수

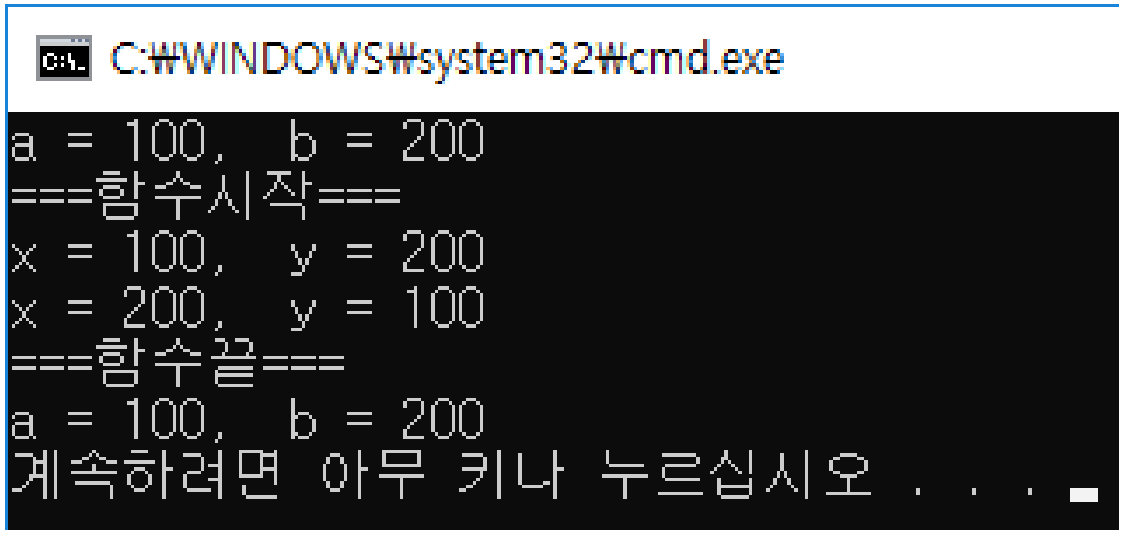
```c
#include<stdio.h>
#include<stdlib.h>
#define SIZE 6
void square_array(int a[], int size) {
    for (int i = 0; i < size; i++)
        a[i] = a[i] * a[i];
}
void print_array(int a[], int size) {
    for (int i = 0; i < size; i++)
        printf("%3d ", a[i]);
    printf("\n");
}
int main() {
    int list[SIZE] = { 1,2,3,4,5,6 };

    print_array(list, SIZE);
    square_array(list, SIZE);
    print_array(list, SIZE);
    return 0;
}
```



함수에서 제곱하고 출력해보았더니 원본이 변경되어 있는 것을 볼 수 있다.

포인터와 함수

```c
#include<stdio.h>

void swap(int x, int y) {
    int tmp;
    printf("===함수시작===\nx = %d,  y = %d\n", x, y);
    tmp = x;
    x = y;
    y = tmp;
    printf("x = %d,  y = %d\n===함수끝===\n", x, y);
}

int main() {
    int a = 100, b = 200;

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

    return 0;
}
```



일반 변수를 인수로 넘기면 값에 의한 호출을 하므로 원본은 변경되지 않음.

포인터와 함수

```c
#include<stdio.h>

void swap(int *px, int *py) {
    int tmp;
    printf("===함수시작===\nx = %d,  y = %d\n", *px, *py);
    tmp = *px;
    *px = *py;
    *py = tmp;
    printf("x = %d,  y = %d\n===함수끝===\n", *px, *py);
}

int main() {
    int a = 100, b = 200;

    printf("a = %d,  b = %d\n", a, b);
    swap(&a, &b);//변수가 아닌 주소를 넘김!!!!
    printf("a = %d,  b = %d\n", a, b);

    return 0;
}
```

```
C:₩WINDOWS₩system32₩cmd.exe
a = 100,  b = 200
===함수시작===
x = 100,  y = 200
x = 200,  y = 100
===함수끝===
a = 200,  b = 100
계속하려면 아무 키나 누르십시오 . . .
```

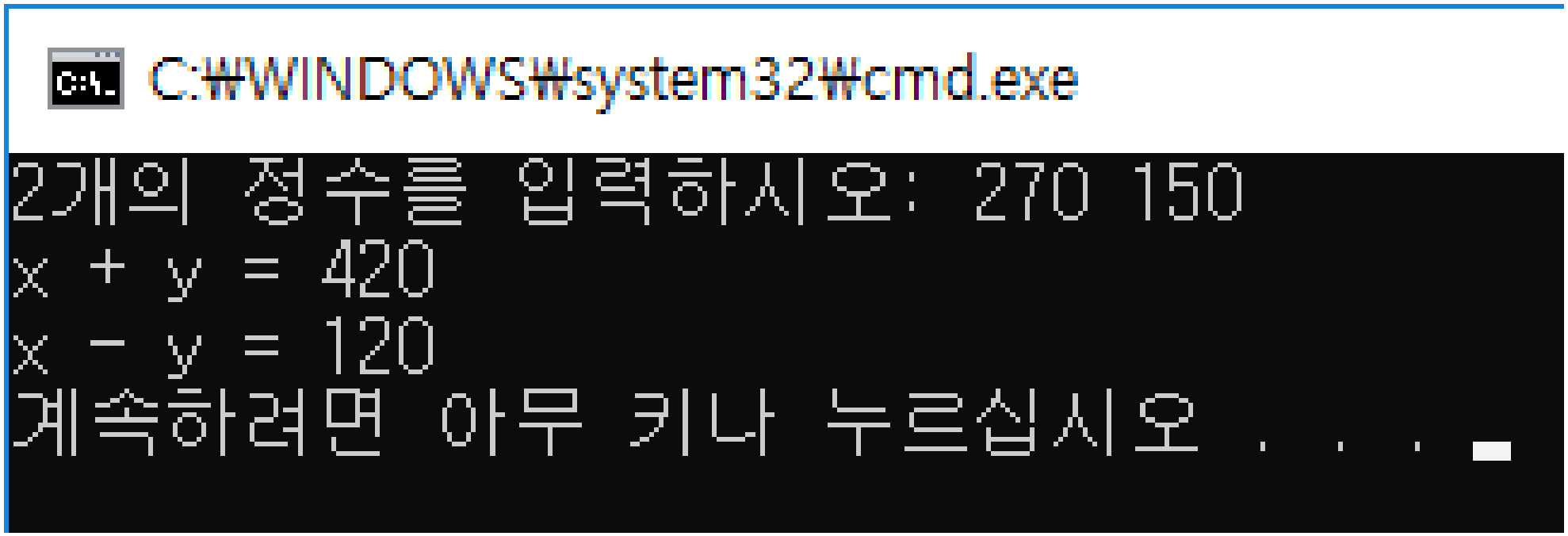
포인터를 인수로 사용하여서 참조에 의한 호출을 이용함. 원본이 변경됨을 확인할 수 있음.

포인터의 장점

1. 포인터를 이용하여 연결 리스트나 트리 등의 향상된 자료구조 구현
2. 참조에 의한 호출
3. 동적 메모리 할당

실습 문제

- 2개의 정수를 입력받는 함수 get_int와 2개의 정수의 합과 차를 동시에 반환하는 함수 get_sum_diff를 작성하라. 포인터 매개 변수를 이용한다.



실습 문제

```c
#include<stdio.h>
void get_int(int* x, int* y);
void get_sum_diff(int x, int y, int* sum, int* diff);

int main(void)
{
    int x, y;
    int sum, diff;

    get_int(&x, &y);
    get_sum_diff(x, y, &sum, &diff);

    printf("x + y = %d\n", sum);
    printf("x - y = %d\n", diff);

    return 0;
}
```

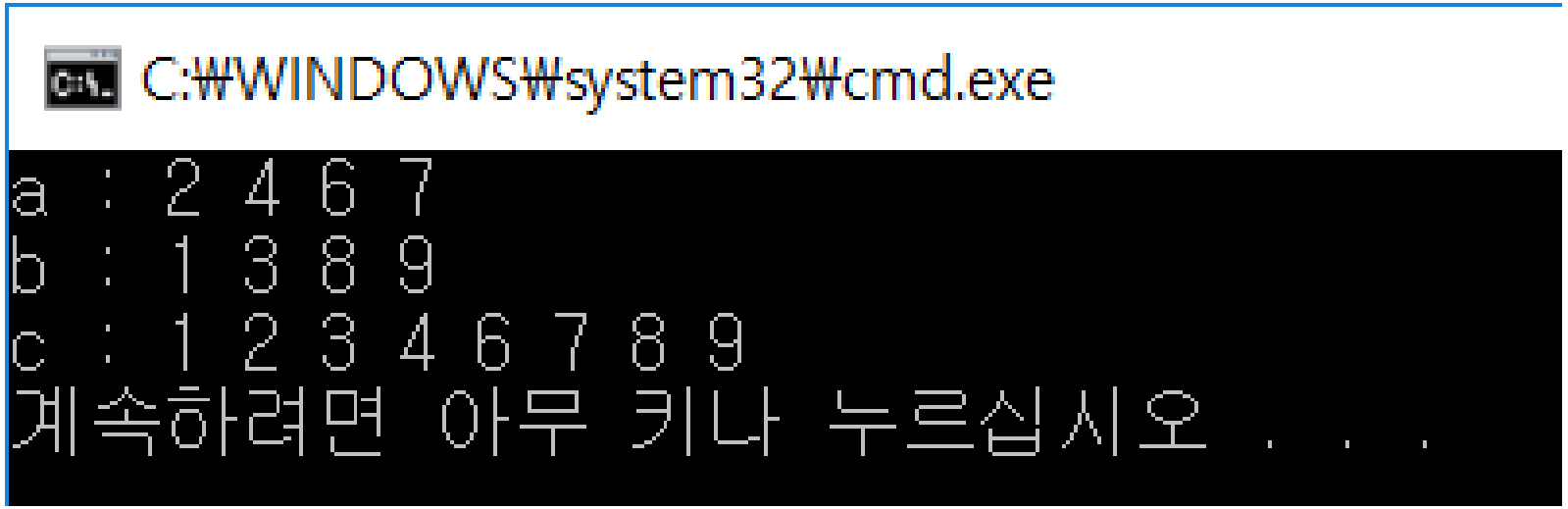
```c
void get_int(int* x, int* y)
{
    printf("두 개의 정수를 입력하시오 : ");
    scanf("%d %d", x, y);
}
```

```c
void get_sum_diff(int x, int y, int* sum, int* diff)
{
    *sum = x + y;
    *diff = x - y;
}
```

실습 문제

- 2개의 정렬된 배열 a, b가 있다. a, b 배열의 크기는 같다.
- 이 두 배열과 크기를 매개변수로 받아서 하나의 배열로 합치는 merge 함수를 작성하여라.

  (새로운 배열 c도 매개변수로 받는다. c의 공간은 넉넉하게 존재한다고 가정한다.)



실습 문제

```c
#include<stdio.h>
void merge(int* a, int* b, int* c, int length);

int main(void)
{
    int i;
    int a[] = { 2, 4, 6, 7 };
    int b[] = { 1, 3, 8, 9 };
    int c[8];

    merge(a, b, c, 4);
   printf("a : ");
   for (i = 0;i < 4;i++)
       printf("%d ", a[i]);
   printf("\n");

   printf("b : ");
   for (i = 0;i < 4;i++)
       printf("%d ", b[i]);
   printf("\n");

   printf("c : ");
   for (i = 0;i < 8;i++)
       printf("%d ", c[i]);
   printf("\n");
}
```

```c
void merge(int* a, int* b, int* c, int length)
{
    int i = 0, j = 0;
    int k = 0;

    while (i < length && j < length)
    {
        if (a[i] > b[j])
            c[k++] = b[j++];
        else
            c[k++] = a[i++];
    }

    while (i < length)
        c[k++] = a[i++];

    while (j < length)
        c[k++] = b[j++];
}
```

다음주?

문자열, 구조체, 동적할당

감사합니다