

# 포인터(Pointer)



복습

포인터 변수

다양한 포인터 변수

포인터 사용 시 주의 사항

## [복습] 다음 코드의 결과를 생각해보고 실행해보자

```
#include <stdio.h>
void Swap(int x, int y);

int main(void)
{
    int a = 10;
    int b = 20;

    printf("Swap 전의 a = %d, b = %d\n", a, b);
    Swap(a, b);
    printf("Swap 후의 a = %d, b = %d\n", a, b);

    return 0;
}

void Swap(int x, int y)
{
    int temp;

    temp = x;
    x = y;
    y = temp;
}
```

### 실행 결과

```
Swap 전의 a = 10, b = 20
Swap 후의 a = 10, b = 20
```

두 변수의 값이  
맞교환되지 않았다.

[연습] 다음 swap 함수를 전역 변수를 사용하여 다시 코딩하시오.

```
#include <stdio.h>
void Swap(int x, int y);

int main(void)
{
    int a = 10;
    int b = 20;

    printf("Swap 전의 a = %d, b = %d\n", a, b);
    Swap(a, b);
    printf("Swap 후의 a = %d, b = %d\n", a, b);

    return 0;
}

void Swap(int x, int y)
{
    int temp;

    temp = x;
    x = y;
    y = temp;
}
```

[코드] 다음 swap 함수를 전역 변수를 사용하여 다시 코딩하시오

```
#include <stdio.h>
void Swap( );
int a=10;
int b=20;

int main(void)
{
    printf("Swap 전의 a = %d, b = %d\n", a, b);
    Swap( );
    printf("Swap 후의 a = %d, b = %d\n", a, b);

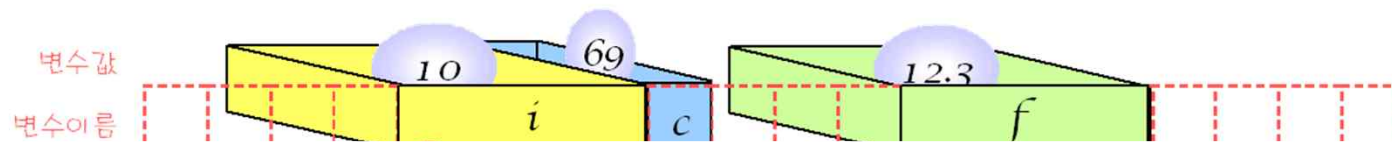
    return 0;
}

void Swap( )
{
    int temp;

    temp = a;
    a = b;
    b = temp;
}
```

# 변수

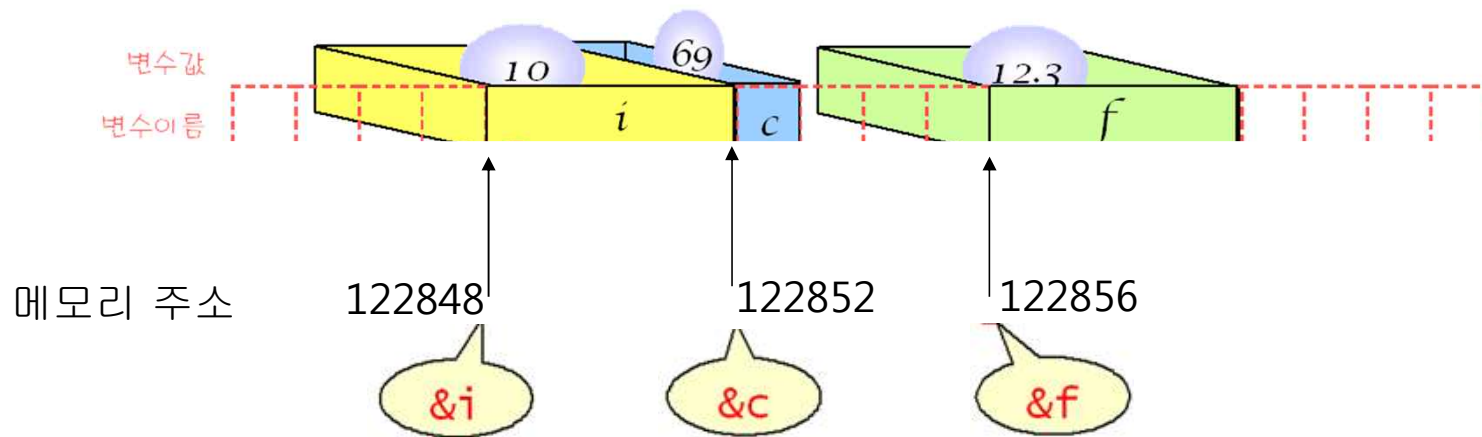
```
int i = 10;
char c = 69;
float f = 12.3;
```



- 변수는 그 종류에 따라서 차지하는 메모리 공간이 달라진다.
- char형 변수: 1바이트, int형 변수: 4바이트, float형 변수: 4바이트,...

## 변수와 메모리 주소

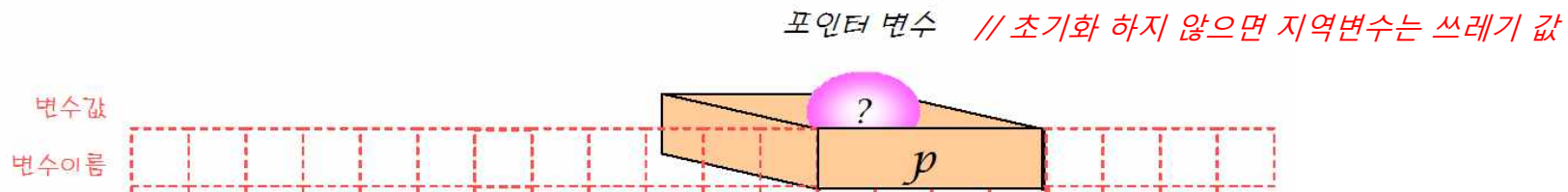
```
int i = 10;
char c = 69;
float f = 12.3;
```



- 변수는 메모리에 저장되며,  
그 주소를 address operator &를 통해 알려줄 수 있다  
예) 변수 i의 주소: &i

## 포인터 변수(pointer variable)

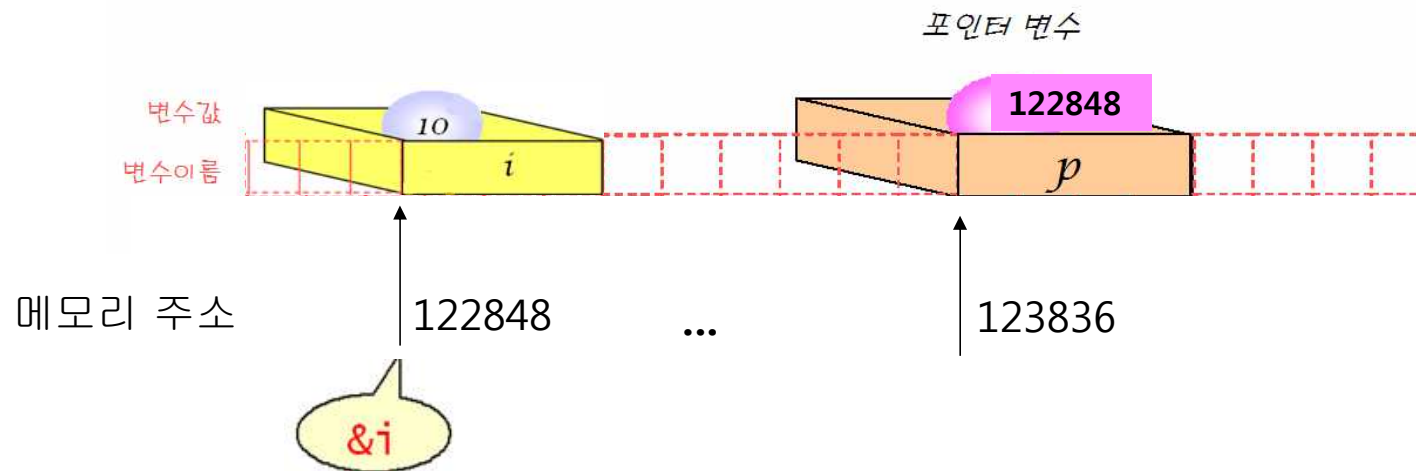
- 변수의 메모리 주소를 저장하기 위해서는 일반적인 변수가 아닌 포인터 변수를 활용
- 포인터 변수의 선언  
예) `int * p;`





```
int i = 10;
int *p;      // 포인터 변수 p 선언
              // 다른 변수의 메모리 주소를 가지는 변수

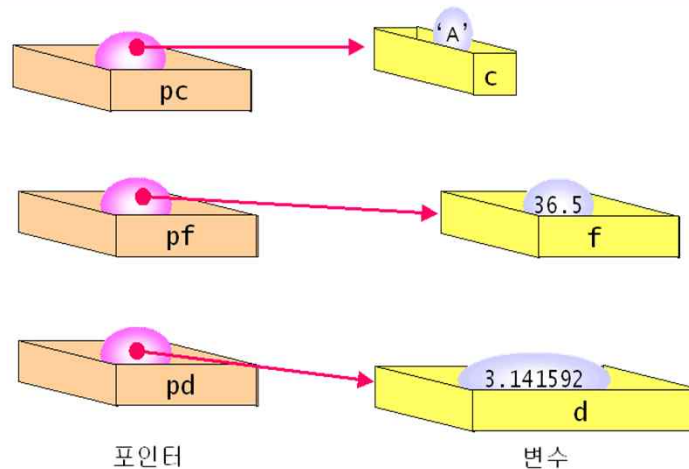
P = &i;      // p는 변수 i의 메모리 주소 122848을 가짐
```



## 다양한 포인터 변수의 선언

```
char c = 'A';           // 문자형 변수 c
float f = 36.5f;        // 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
```



**포인터 변수의 크기**는 포인터 변수가 가리키는 변수의 데이터 형에 관계없이 **4바이트로 항상 같다**

[연습] 다음의 코드 진행 과정을 메모리 그림으로 그려서 결과를 예측하고, 실행 후 비교하시오

```
#include <stdio.h>
int main(void)
{
    int i = 10;
    char c = 69;
    float f = 12.3f;

    int *pi = &i;
    char *pc = &c;
    float *pf = &f;

    printf("i의 주소: %d %d\n", &i, pi); // 변수 i의 주소 출력
    printf("c의 주소: %d %d\n", &c, pc); // 변수 c의 주소 출력
    printf("f의 주소: %d %d\n", &f, pf); // 변수 f의 주소 출력
    return 0;
}
```

i의 주소: 1245024 1245024 //두 주소 값은 동일(각자 실행하면 숫자는 다를 수 있음)  
 c의 주소: 1245015 1245015  
 f의 주소: 1245000 1245000

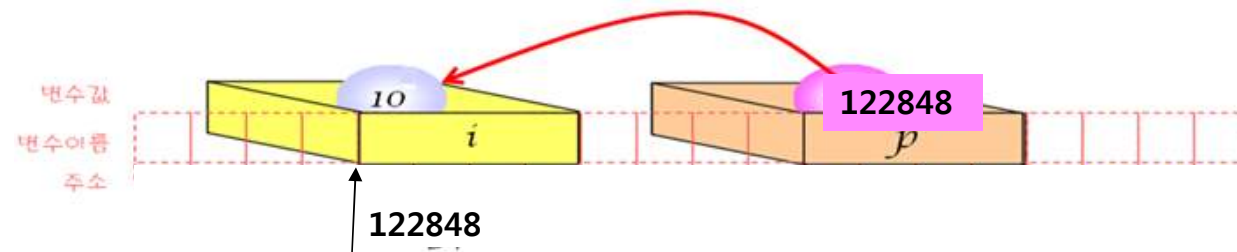
## 간접 참조 \* (Indirection Operator \*)

```
int i = 10;
int *p;
p = &i;
```

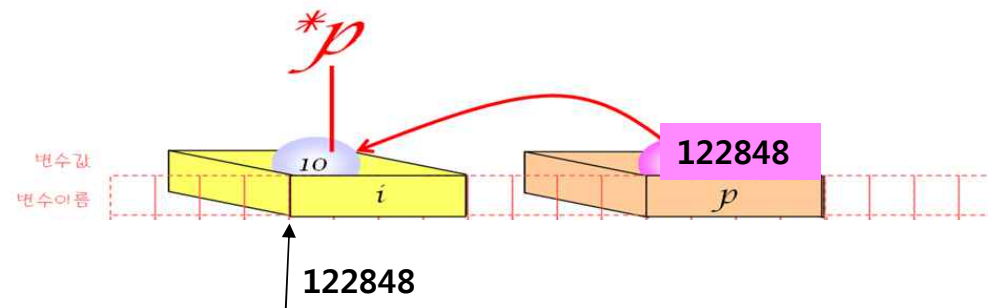
간접 참조 \* (Indirection Operator \*)를 사용

-포인터가 가리키는 곳의 값을 포인터 타입에 따라 가져옴

예) printf("%d", \*p); //10



포인터 p를 통해 p가 가리키는 변수 i의 값인 10을 가져오려면?



```
int i=10;  
int *p = &i;
```

```
printf("%d", p); //&i와 동일  
printf("%d", *p); //10
```

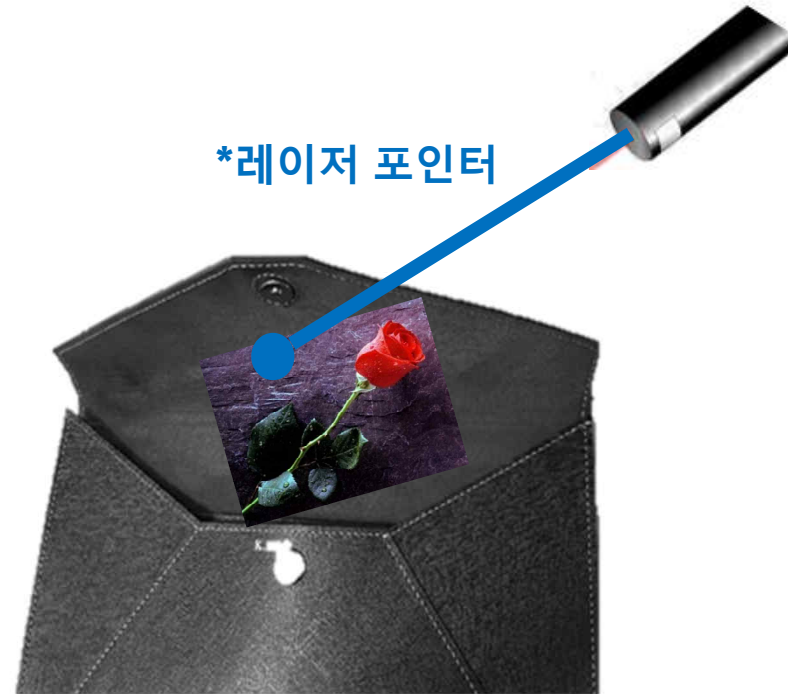
봉투형 가방 <= 장미꽃 그림엽서  
레이저 포인터 <= 봉투형 가방이 있는 위치

**레이저 포인터** 출력 //봉투형 가방이 있는 위치  
**\*레이저 포인터** 출력 // 장미꽃 그림엽서

레이저 포인터



\*레이저 포인터

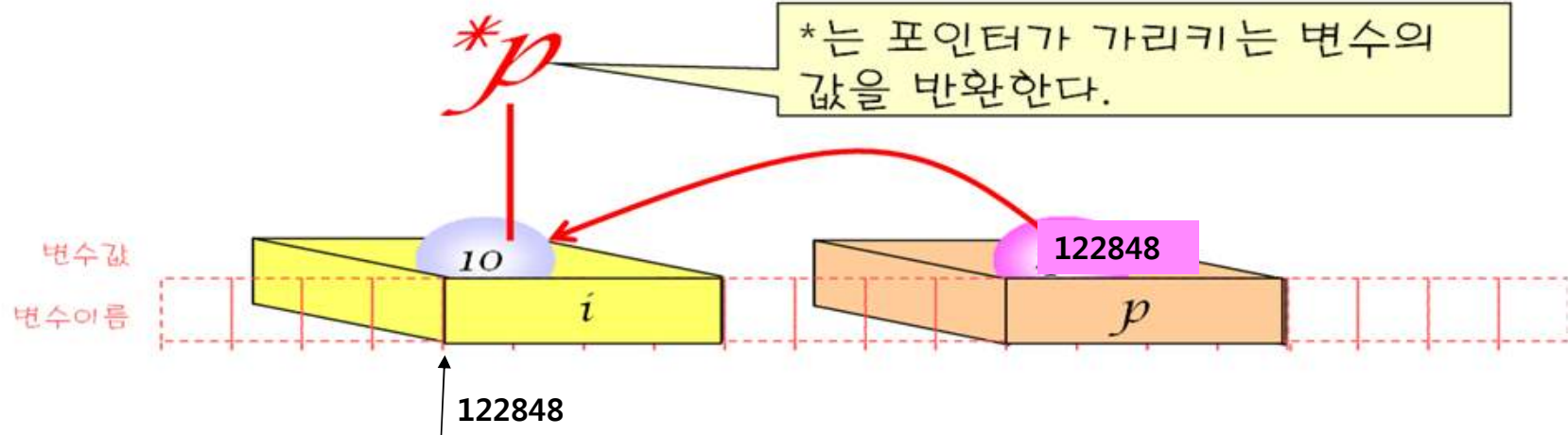


```
int i=10;
int *p;

p = &i;
printf("%d", *p); //10
```

### [ \* 의 용도 ]

2 \* 3 → 곱셈  
 int \*p → 포인터 변수 선언  
 \*p → 포인터가 가리키는 곳의 값



[실습] 다음의 코드 진행 과정을 메모리 그림으로 그려서 결과를 예측하고,  
실행 후 비교하시오

```
#include <stdio.h>

int main(void)
{
    double d = 10.5;
    double *pd = &d;

    *pd = *pd * 3;
    printf("%f %f\n", *pd, d);

    d = *pd - 0.2;
    printf("%f %f\n", *pd, d);
    return 0;
}
```

**[결과]**

31.500000 31.500000  
31.300000 31.300000

## 포인터 사용 시 주의점

- 포인터가 아무것도 가리키고 있지 않는 경우에는 NULL로 초기화
- 포인터의 타입과 변수의 타입은 일치하여야 한다.

```
int i;
double *pd = NULL;

pd = &i; // 경고 및 실행 오류 발생 가능성!
        // double형 포인터에 int형 변수의 주소를 대입

*pd = 36.5;
```

- 상수나 수식 앞에는 주소를 알려주는 &를 사용할 수 없음
  - 잘못 사용한 예
    - &3 // 상수 앞에서 사용
    - &(i + 3) // 수식 앞에서 사용



[연습] 다음 코드의 결과를 예측하고, 실행 후 비교하시오  
또한 코드 진행 과정을 메모리 그림으로 그리시오

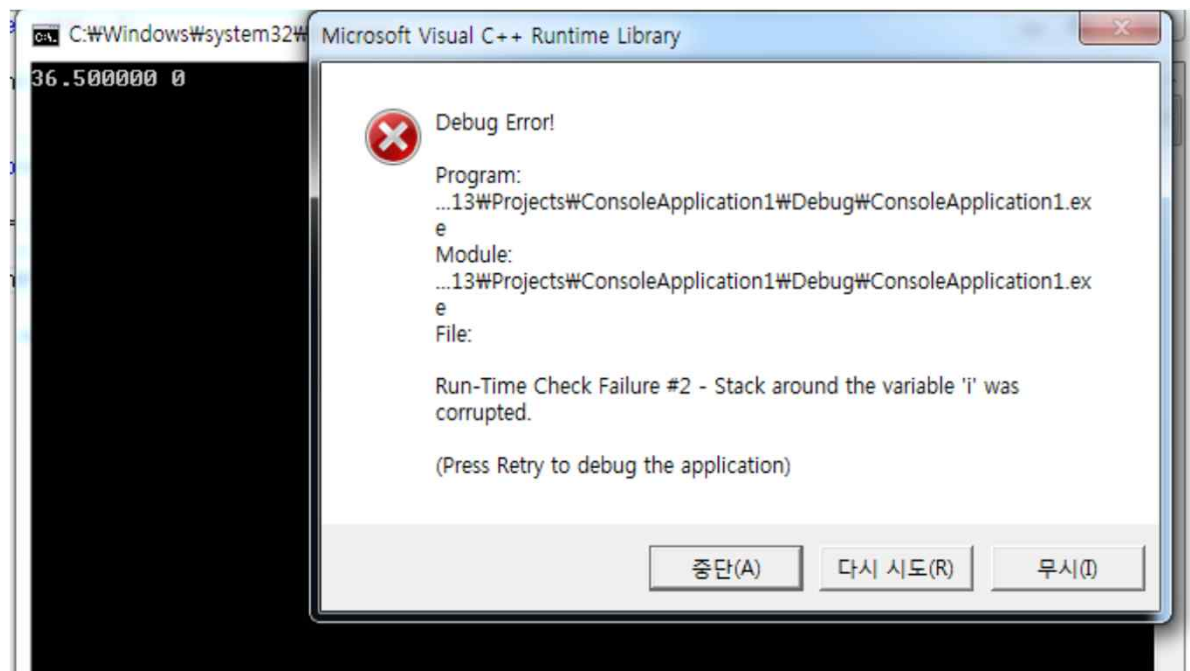
```
#include <stdio.h>

int main(void)
{
    int i = 10;
    double *pd = NULL;

    pd = &i;
    *pd = 36.5;
    printf("%f %d\n", *pd, i);

    return 0;
}
```

## [결과]



## [수정]

```
#include <stdio.h>

int main(void)
{
    double d_num = 12.3;
    double *pd = NULL;

    pd = &d_num;
    *pd = 36.5;
    printf("%f %f\n", *pd, d_num);
    return 0;
}
```

```
#include <stdio.h>

int main(void)
{
    int i = 10;
    int *pi = NULL;

    pi = &i;
    *pi = 36;
    printf("%d %d\n", *pi, i);

    return 0;
}
```

[연습] 다음 swap 함수를 포인터를 사용하여 다시 코딩하시오.  
또한 코드 진행 과정을 메모리 그림으로 그리시오

```
#include <stdio.h>
void Swap(int x, int y);

int main(void)
{
    int a = 10;
    int b = 20;

    printf("Swap 전의 a = %d, b = %d\n", a, b);
    Swap(a, b);
    printf("Swap 후의 a = %d, b = %d\n", a, b);

    return 0;
}

void Swap(int x, int y)
{
    int temp;

    temp = x;
    x = y;
    y = temp;
}
```

```
#include <stdio.h>
void Swap(int *x, int *y);

int main(void)
{
    int a = 10;
    int b = 20;

    printf("Swap 전의 a = %d, b = %d\n", a, b);
    Swap(&a, &b);
    printf("Swap 후의 a = %d, b = %d\n", a, b);

    return 0;
}

void Swap(int *x, int *y)
{
    int temp;

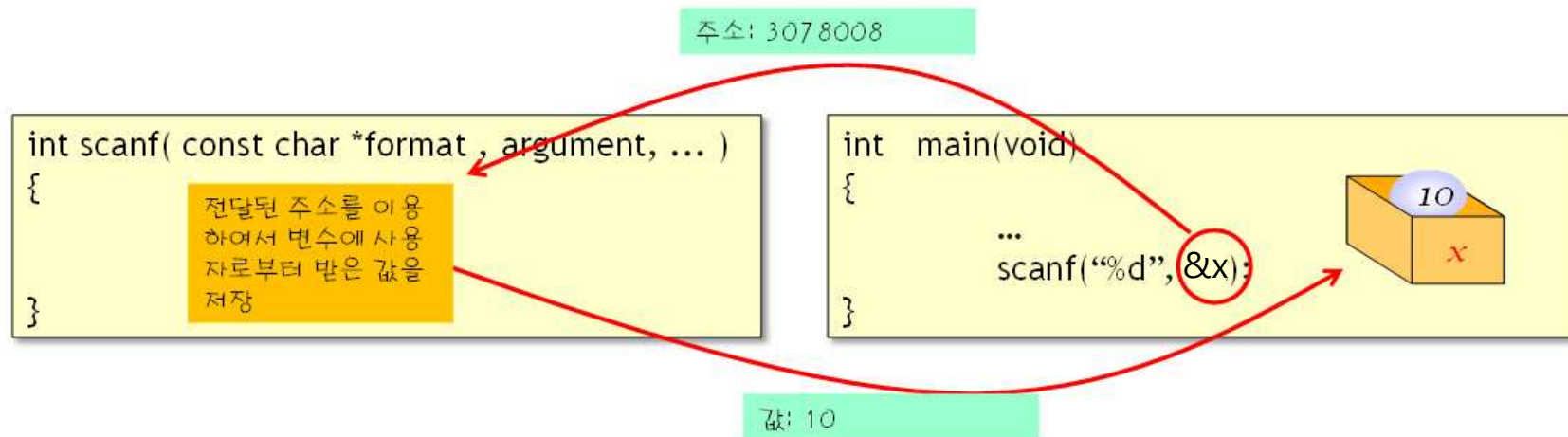
    temp = *x;
    *x = *y;
    *y = temp;
}
```

## 함수 호출시 참조(Call by Reference)에 의한 전달

- Swap 함수처럼  
함수 안에서 2개 이상의 변경된 값을 호출한 함수로 보낼 때에는  
변수 값을 복사해서 전달(Call by Value)하는 대신  
변수 주소를 전달하는 **참조에 의한 전달 방법(Call by Reference)**을 사용할 수 있음

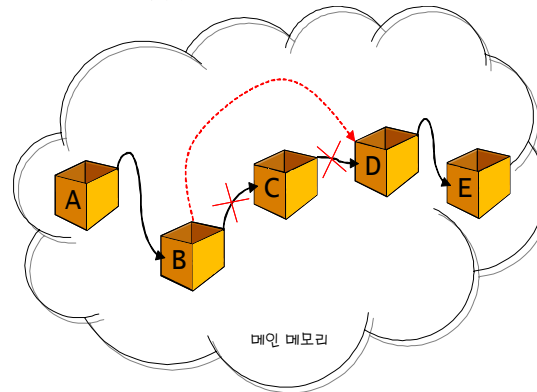
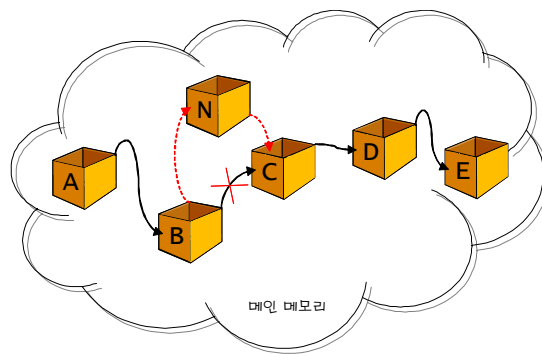
## scanf() 함수와 포인터의 활용

- 변수에 값을 저장하기 위하여 변수의 주소를 받는다.



## 포인터 사용의 장점

- 참조에 의한 호출
  - 포인터를 사용하여 함수 외부의 변수 값을 변경할 수 있다
  - 함수에서 배열을 전달 시, 기억 장소 및 원소 값 복사 시간의 낭비를 줄임  
(차후에 배움)
- 동적 메모리 할당(dynamic memory allocation)
  - 요소의 개수가 미리 결정되어 사용하는 배열과 달리, 필요할 때마다 메모리를 할당하여 사용하여 융통성 있는 프로그램 작성 가능
  - 연결 리스트나 이진 트리 등의 향상된 자료 구조를 만들 수 있다





## [실습] 다음 조건대로 코딩하시오 또한 코드 진행 과정을 메모리 그림으로 그리시오

$$5 + 8 = 13$$

$$5 * 8 = 40$$

- main 함수로부터 두 정수를 받아 그 합과 곱의 결과를 계산하고,  
main 함수에서 그 결과를 출력할 수 있도록 하는 **사용자 정의 함수**를 포함한  
C 프로그램을 작성하시오  
(단, 전역 변수는 사용하지 않으며,  
사용자가 종료를 원할 때까지 반복함.  
그 외는 임의로 조건을 설정하되, 한 줄 주석으로 처리할 것 )

```
#include <stdio.h>

void GetSumProduct(int x, int y, int *sum, int *product);

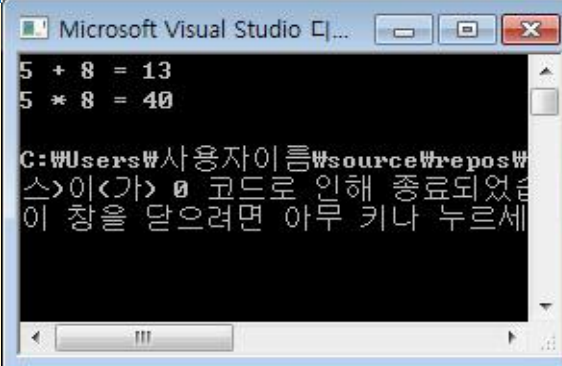
int main(void)
{
    int a = 5, b = 8;
    int res1, res2;

    GetSumProduct(a, b, &res1, &res2);

    printf("%d + %d = %d\n", a, b, res1);
    printf("%d * %d = %d\n", a, b, res2);

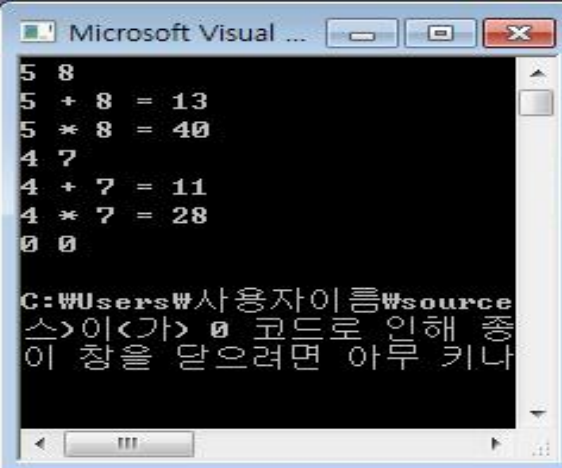
    return 0;
}

void GetSumProduct(int x, int y, int *sum, int *product)
{
    *sum = x + y;
    *product = x * y;
}
```



```
Microsoft Visual Studio 디...
5 + 8 = 13
5 * 8 = 40
C:\Users\사용자이름\source\repos\스>이<가> 0 코드로 인해 종료되었습
이 창을 닫으려면 아무 키나 누르세
```

//결과가 이와 같이 되도록 수정



```
Microsoft Visual ...
5 8
5 + 8 = 13
5 * 8 = 40
4 7
4 + 7 = 11
4 * 7 = 28
0 0
C:\Users\사용자이름\source\스>이<가> 0 코드로 인해 종
이 창을 닫으려면 아무 키나
```

//a, b가 모두 0인 경우, 반복이 종료됨

[실습] 다음과 같은 결과가 출력되도록 나눗셈 함수를 정의하여 코딩하시오  
또한 코드 진행 과정을 메모리 그림으로 그리시오

나눗셈을 위한 분자를 입력하세요 : 541  
나눗셈을 위한 분모를 입력하세요 : 99  
541 / 99 : 몫은 5이고 나머지는 46입니다.

나눗셈을 위한 분자를 입력하세요 : 0  
나눗셈을 위한 분모를 입력하세요 : 99  
0 / 99 : 몫은 0이고 나머지는 0입니다.

나눗셈을 위한 분자를 입력하세요 : 102  
나눗셈을 위한 분모를 입력하세요 : 0  
0으로 나눌 수 없습니다.

### [조건]

모든 입출력은 main함수에서 이루어지며,  
나눗셈 **사용자 정의 함수**는 계산만 처리한다.

(단, 전역 변수는 사용하지 않으며,  
사용자가 메뉴 선택 및 종료를 원할 때까지 반복함.  
그 외는 임의로 조건을 설정하되, 한 줄 주석으로 처리할 것 )



 T h a n k y o u

## TECHNOLOGY

>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Velit ex  
aliqua ipsum, labore sed tempore ratione asperiores des  
consectetuer tempore rati igert one bore sed teni