

Robotics



대경혁신인재양성프로젝트
HuStar

C프로그래밍5

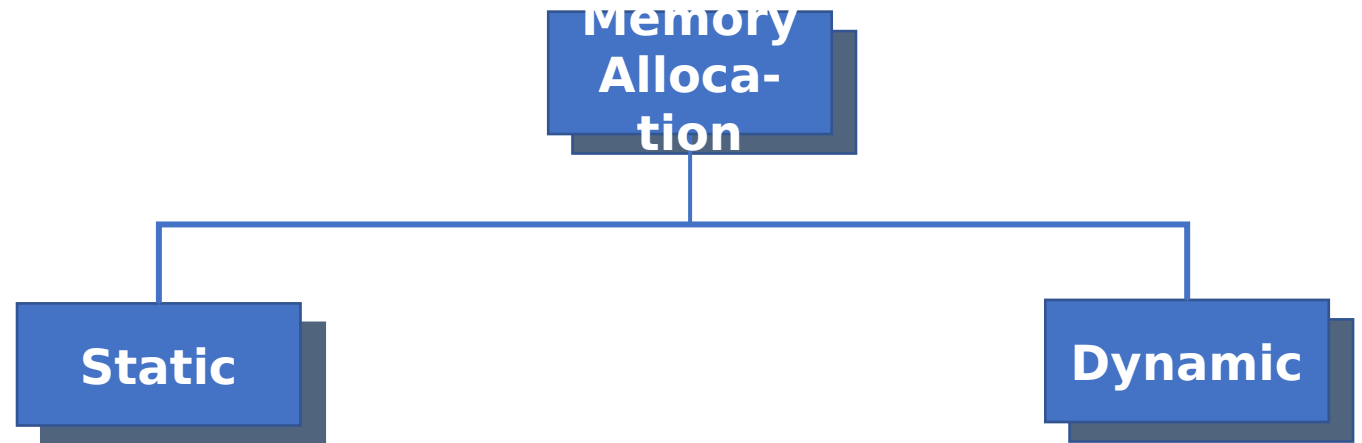
Robotics



메모리 할당

동적 메모리 할당

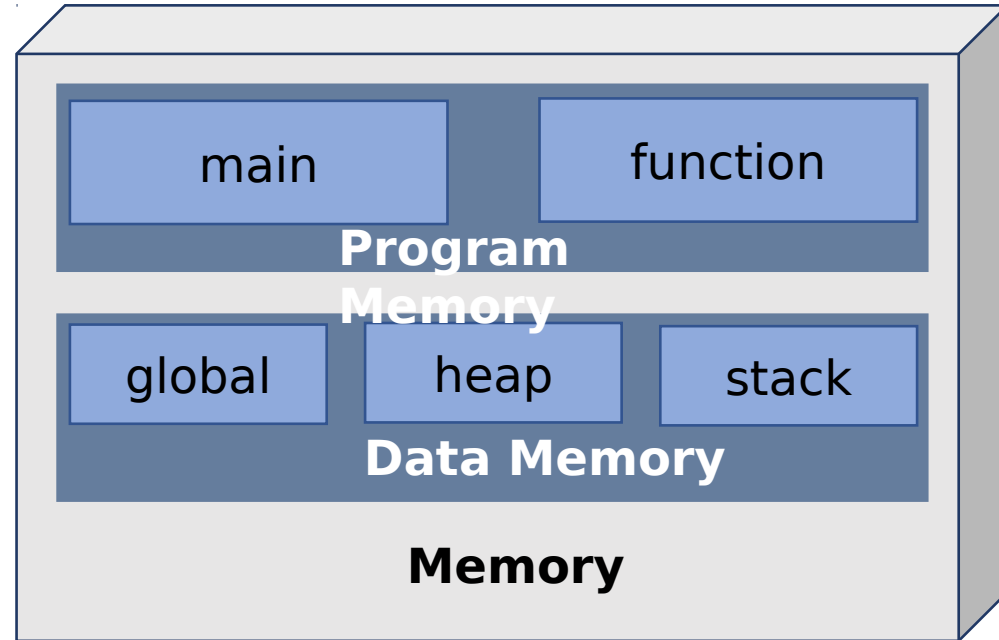
- 메모리 할당
 - 프로그램이 실행되는 동안 메모리 저장 공간을 할당 하는 것
- 메모리 할당의 방식
 - 정적 할당(static allocation)
 - 동적 할당(dynamic allocation)



동적 메모리 할당(dynamic memory allocation)

- 메모리의 개념

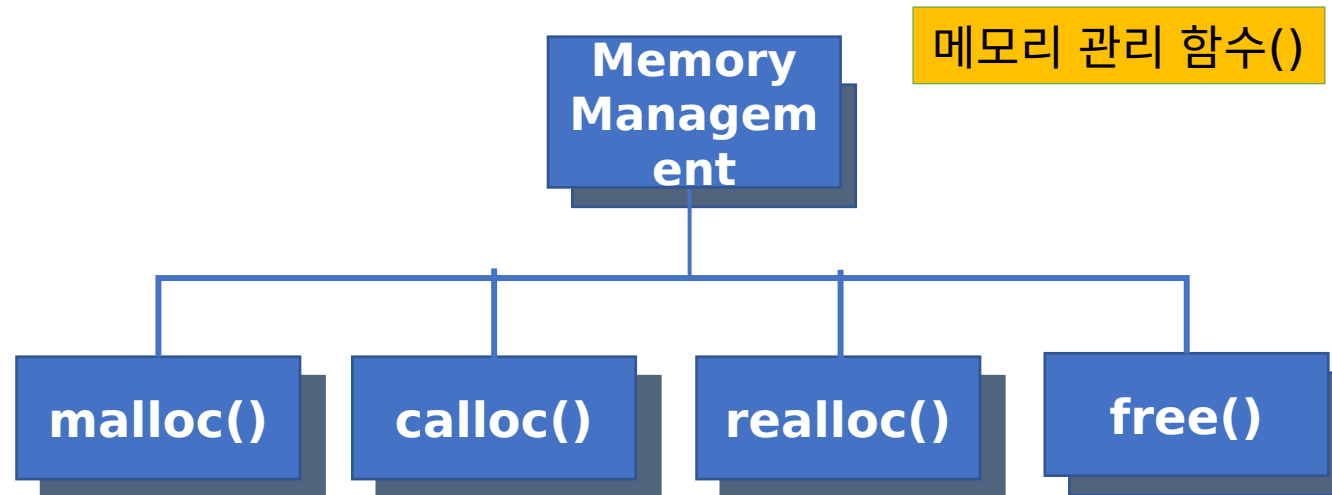
- Program memory: 프로그램 코드(명령어의 집합)
- Data memory: 저장 공간(변수, 동적 메모리)
 - Global memory: 전역 변수
 - Heap: 동적으로 할당되는 메모리
 - stack: 지역 변수



Heap에서 메모리 할당은 오직
포인터를 통해서만 가능

동적 메모리 할당(dynamic memory allocation)

- 정적 저장공간
 - 프로그램에서 선언된 저장공간
 - Stack(지역변수) 또는 전역 데이터 메모리(전역 변수)에서 할당됨
- 동적 저장공간
 - 프로그램이 실행되는 동안 할당되는 메모리 저장 공간



동적 메모리 할당(dynamic memory allocation)

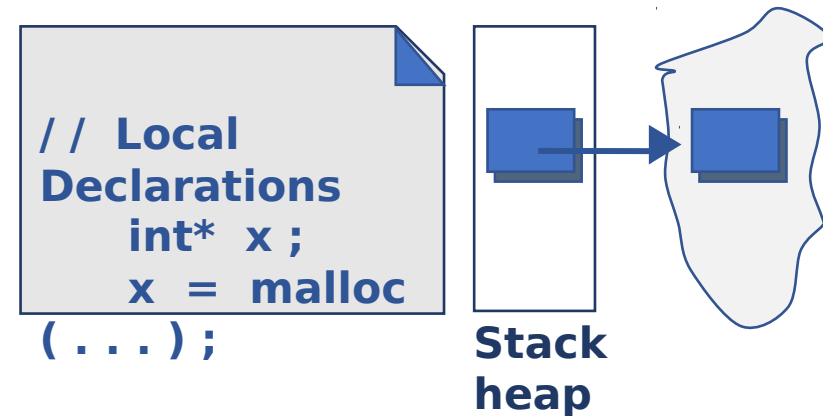
- 메모리에 접근 방법

- 정적 메모리 할당



- 동적 메모리 할당

메모리 할당은 미리 정의된 함수를 이용
저장공간 크기는 동적으로 결정됨
Heap의 공간으로부터 할당되어 짐



동적 메모리 할당(dynamic memory allocation)

- 예제

//정적 할당

```
int main()
{
    int n = 0, i = 0;
    int array[100];
    int *data = array;

    printf("Input # of data:");
    scanf("%d", &n);

    for(i = 0; i < n; i++)
        scanf("%d", &data[i]);
    ...
}
```

//동적 할당

```
int main()
{
    int n = 0, i = 0;
    int *data = NULL;

    printf("Input # of data:");
    scanf("%d", &n);

    data = (int*)malloc(n*sizeof(int))
    for(i = 0; i < n; i++)
        scanf("%d", &data[i]);
    ...
    free(data);
}
```

동적 메모리 할당(dynamic memory allocation)

- 메모리 할당 함수
 - 라이브러리 함수(함수의 prototype은 <stdlib.h>에 정의되어 있음)

Malloc() 함수

byte 단위의 요청된 크기로 할당

할당된 저장공간에서 첫번째 byte에 대한 포인터를 반환(return)

calloc() function

배열의 요소를 위한 공간을 할당

메모리에 대한 포인터 반환(return)

0으로 초기화

realloc() function

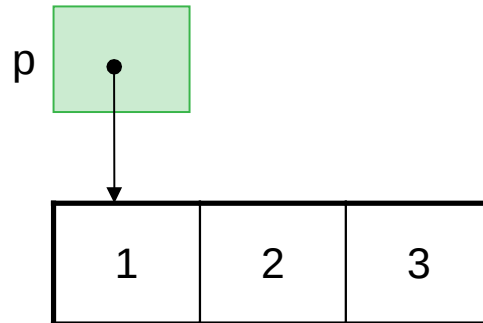
삭제 또는 확장으로 메모리 블록의 크기를 변화 시킴

동적 메모리 할당(dynamic memory allocation)

- malloc()의 사용

```
void *malloc(size_t number_of_bytes);  
/* void* 형식의 값을 반환-일반 포인터- 메모리주소. */  
/* size_t : <stdlib.h>에 unsigned int형으로 정의되어있음  
*/
```

```
int *p;  
p=(int*)  
malloc(12);  
p[0]=1;  
p[1]=2;  
p[2]=3;
```

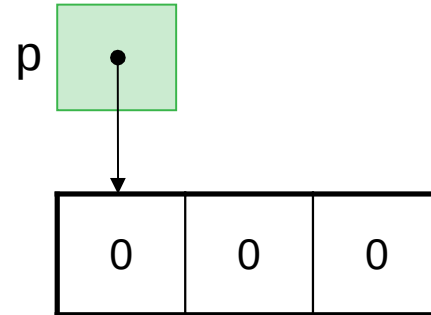


동적 메모리 할당(dynamic memory allocation)

- Calloc() 함수 사용

```
void *calloc(size_t number_of_element, size_t element_size);  
/* number of elements : 요소의 수  element_size : byte단위로 지정된 요소의 크기 */  
/* automatically initialized*/
```

```
int *p;  
p=(int*) calloc(3,  
sizeof(int));  
  
// Same as p=(int*)  
calloc(3,4);
```

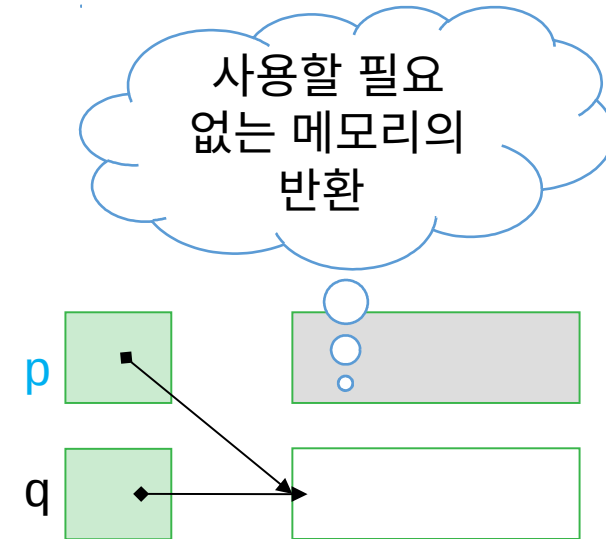


동적 메모리 할당(dynamic memory allocation)

- Free()
 - 할당된 메모리를 해제

```
void free(void *ptr);
```

```
p = malloc(...);  
q = malloc(...);  
free(p); // free() 함수호출은 p 포인터에서  
  
// 가리키고 있는 메모리 블록을 해제 한다  
p = q;
```



동적 메모리 할당(dynamic memory allocation)

- 예제

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int *a, i, n, sum = 0;
    printf("\n%s", "An array will be created dynamically.\n\n");
    printf("Input an array size n followed by n integers: ");
    scanf("%d", &n );

    a = calloc(n, sizeof(int) ); //or a = malloc(n * sizeof(int) );
    for ( i = 0; i < n; ++i )    scanf("%d", &a[ i ] );
    for ( i = 0; i < n; ++i )    sum += a[ i ];
    free(a);
    printf("\n%s%7d\n%s%7d\n\n", "Number of elements: ", n, "Sum of the elements: ", sum );
}
```

Robotics



Derived types

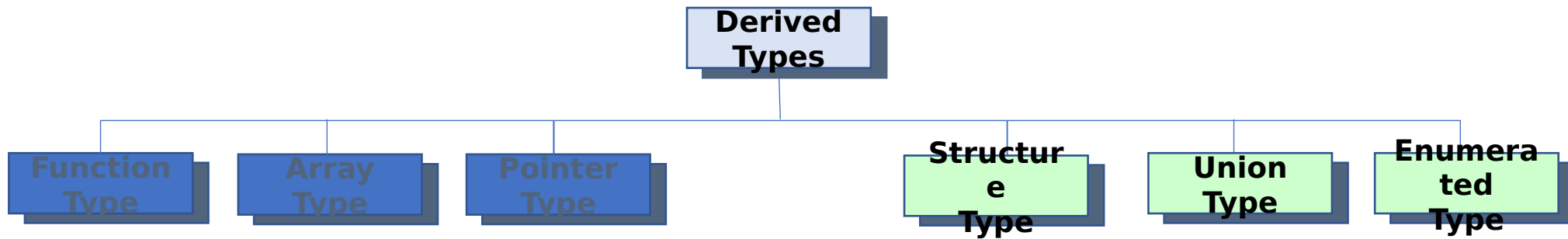
파생 데이터 타입

기본형

char, short, int, long, long long
float, double float, long double

파생형

Structure, Enum, union



자료 형식 생성(type definition)

- Typedef

- 기존의 데이터 형식에 새로운 이름을 부여하는 역할
- 프로그램의 이해도와 수정을 쉽게 도와주는 역할

typedef type IDENTIFIER;

C언어에서 지원하는
데이터 형식 또는
파생된 데이터 형식

사용자가 지정 하고
싶은 이름

```
typedef int Bool ;  
typedef long int cash ;  
  
Bool flag ;  
cash cash_in, cash_out  
;
```

```
typedef int ptrdiff_t ;  
  
typedef unsigned size_t  
;  
  
typedef char* string ;
```

Typedef 예시

```
#include <stdio.h>

typedef int INT;
typedef int * PTR_INT;

typedef unsigned int UINT;
typedef unsigned int *PTR_UINT;

typedef unsigned char UCHAR;
typedef unsigned char *PTR_UCHAR;

int main(){
    INT num1=120;
    PTR_INT pnum1 = &num1;

    UINT num2=190;
    PTR_UINT pnum2 =&num2;

    UCHAR ch='Z';
    PTR_UCHAR pch = &ch;

    printf("%d %u, %c\n", *pnum1, *pnum2, *pch);
    return 0;
}
```

새로운 이름으로
데이터형식을 정의

Enumerated type

- 열거형(enumerated type)
 - 사용자 정의 타입(정수형을 기반)
 - 열거형에 정의된 데이터에 정수값이 주어짐
 - 열거형 상수

```
enum typeName {identifier list};
```

사용자가
데이터형식의
이름을 지정

데이터 정의 목록을
작성

정수가 할당되어 있지 않다면 자동으로 0
부터 1씩 증가하는 형태로 결정

```
enum color {RED, BLUE, GREEN, WHITE};           //define
enum color skyColor;                             //
declare a variable skyColor
```

Syllable에 저장 가능한 정수
값들을 결정하겠다는 의미

예시

```
enum syllable{
    Do=1, Re=2, Mi=3, Fa=4, So=5, La=6, T
    i=7
};
```

Enumerated type

- 다양한 초기화

```
enum months {JAN, FEB, MAR, APR};
```

```
enum months {JAN = 1, FEB, MAR, APR}; //1,2,3,4
```

```
enum color {RED, ROSE = 0, CRIMSON = 0, BLUE, AQUA = 1};
```

```
/* RED, ROSE 와 CRIMSON은 상수 0: BLUE와 AQUA는 상수 1 */
```

#define과 enum의 비교

- Enum을 이용한 정의

```
enum suit{CLUBS, DIAMONDS, HEARTS, SPADES} s1, s2;
```

```
s1 = HEARTS;  
s2 = DIAMONDS;
```

integer 0 integer 1 integer 2 integer 3

- Define 구문을 이용한 정의

```
#define SUIT      int  
#define CLUBS    0  
#define DIAMONDS 1  
#define HEARTS    2  
#define SPADES    3
```

```
SUIT s1, s2;  
s1 = HEARTS;  
s2 = DIAMONDS;
```

Enumerate data type

- typedef를 이용하여 데이터 형식의 정의

Enumeration Tags

```
enum suit {CLUBS, DIAMONDS, HEARTS, SPADES} ;  
enum suit s1, s2 ;
```

열거형 태그

Typedef 이용한 정의

```
typedef enum {CLUBS, DIAMONDS, HEARTS, SPADES} SUIT  
;  
SUIT s1, s2 ;
```

enum data type을 SUIT
설정

Enumeration tag상태에서는 데이터형식으로
사용할 수 없음

Typedef를 이용하여 새로운 이름을 부여하여
데이터형식으로 사용

Enumerated type 예시1

```
int main() {  
    enum TV { SBS=6, KBS2=7, KBS=9, MBC=11} ;  
    enum TV ch1, ch2 ;  
  
    ch1 = SBS + 1;  
    ch2 = MBC ;  
  
    printf("Here are TV channel information \n") ;  
    printf("    SBS : %2d\n", SBS);  
    printf("    KBS2 : %2d\n", ch1);  
    printf("    KBS : %2d\n", KBS);  
    printf("    MBC : %2d\n", ch2);}
```

```
Here are TV channel information  
SBS : 6  
KBS2 : 7  
KBS : 9  
MBC : 11
```

Enumerated type 예시2

```
#include <stdio.h>

typedef enum syllable{
    Do=1, Re=2, Mi=3, Fa=4, So=5, La=6, Ti=7
}Syllable;

void Sound(Syllable sy){
    switch(sy){
        case Do:
            puts("도는 하얀 도라지\n"); return;
        case Re:
            puts("레는 둥근 레코드\n"); return;
        case Mi:
            puts("미는 파란 미나리\n"); return;
        case Fa:
            puts("파는 예쁜 파랑새\n"); return;
        case So:
            puts("솔은 작은 솔방울\n"); return;
        case La:
            puts("라는 라디오고요~\n"); return;
        case Ti:
            puts("시는 졸졸 시냇물\n"); return;
    }

    puts("다 함께 부르세~ 도레미파 솔라시도 솔 도~");
}
```

```
int main(){
    Syllable tone;
    for(tone=Do; tone<=Ti; tone+=1)
        Sound(tone);
    return 0;
}
```

도는 하얀 도라지

미는 파란 미나리

파는 예쁜 파랑새

솔은 작은 솔방울

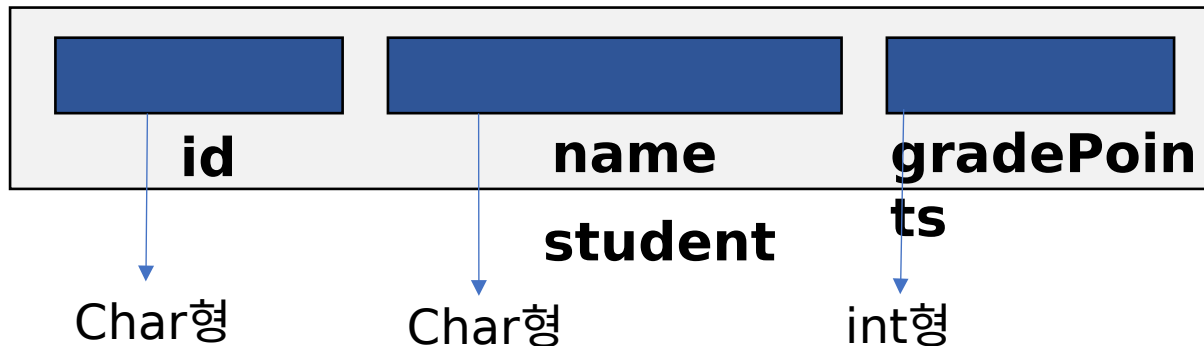
라는 라디오고요~

시는 졸졸 시냇물

구조체(structure)

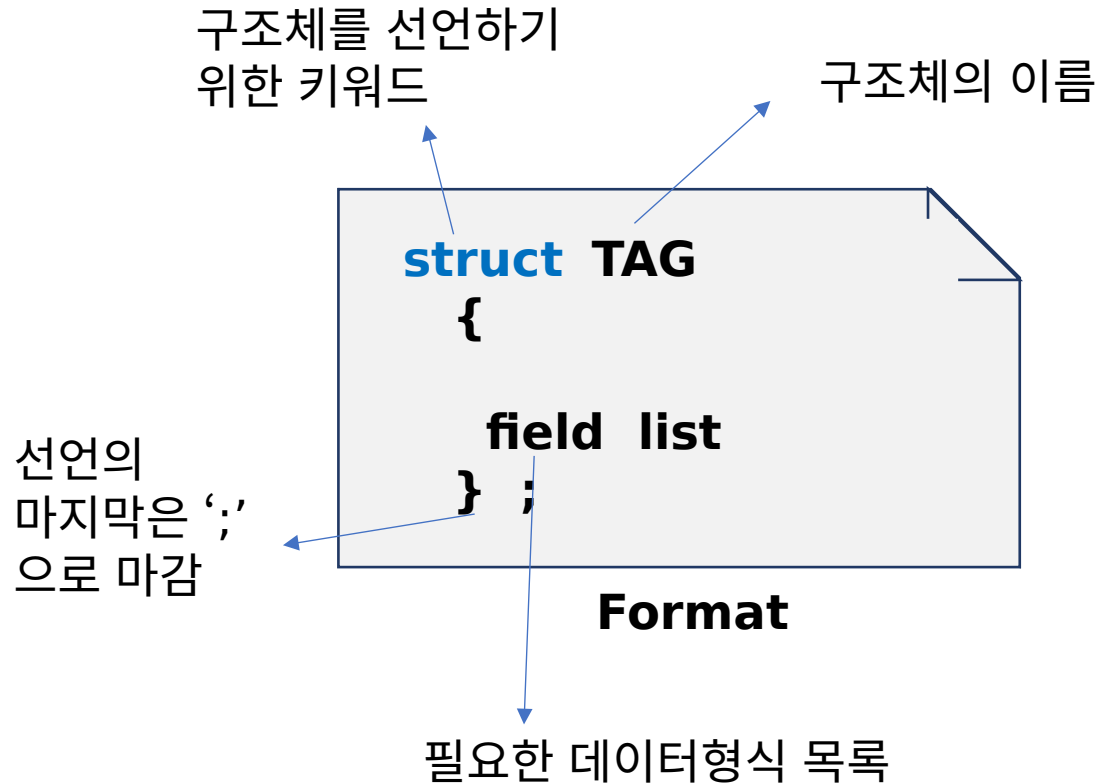
- 구조체:

- 하나 이상의 변수(포인터변수, 배열 포함)를 묶어서 새로운 자료형을 정의하는 도구
- 배열과의 비교
 - 배열은 같은 데이터 형식의 관련된 데이터의 모음
 - Structure는 다른 데이터형식을 하나의 요소로 관리할 수 있음
 - 배열은 파생형 데이터 타입이지만 구조체는 프로그래머가 정의한 형식



3개의 데이터형식이 같이 사용되어야 하는 경우 배열은 불가능함
=> Structure를 이용하여 해결

구조체의 선언



```
struct TAG
{
    char id [ 10 ];
    char name
[ 26 ];
    int gradePts ;
} ; Example
```


구조체의 정의

- syntax `struct tag_name { field-list} variable_identifier`

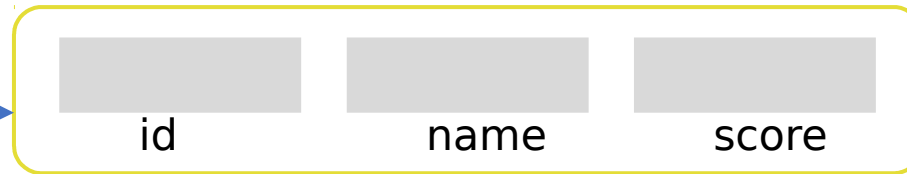
```
struct student{  
    char id[10];  
    char  
    name[20];  
    int score;  
};
```

```
struct student std1;
```

구조체를 생성

...

```
void printstruct(struct  
student);
```



구조체의 정의

```
struct part {  
    int number;  
    char name [20] ;  
    int on_hand;  
} part1, part2;
```

```
struct part part3;
```

part1, part2, part3
모두 사용가능

```
struct part part1, part2 ;    /* correct declaration */  
part part1, part2;           /* wrong declaration */
```

구조체 예제1

- 두 점 사이의 거리 계산

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

struct point{
    int xpos;
    int ypos;
};

int main(){
    struct point pos1, pos2;
    double distance;

    fputs("point1 pos: ", stdout);
    scanf("%d %d", &pos1.xpos, &pos1.ypos);

    fputs("point2 pos: ", stdout);
    scanf("%d %d", &pos2.xpos, &pos2.ypos);

    distance=sqrt((double)((pos1.xpos-pos2.xpos)*(pos1.xpos-
pos2.xpos)+(pos1.ypos-pos2.ypos)*(pos1.ypos-pos2.ypos)));

    printf("distance: %g\n",distance);
    return 0;
}
```

```
point1 pos: 10 20
point2 pos: 30 40
distance: 28.2843
```

구조체의 정의

- Typedef를 이용한 정의
 - 가장 일반적인 방법
 - 형식 정의형 구조체

<syntax>

```
typedef struct {  
    Field list  
    :  
} TAG_NAME;
```

```
typedef struct {  
    char id[10];  
    char name[20];  
    int score;  
} STUDENT;           //definition  
Struct STUDENT std1; //declaration  
...  
  
void printstruct(STUDENT); //function  
prototype
```

구조체의 정의 비교

- 구조체의 정의 및 비교

```
// structure tag
struct STUDENT
{
    char    id [ 10 ];
    char    name [ 26 ];
    int     gradePts ;
} ;

struct STUDENT aStudent
;
```

```
// typedef
typedef struct
{
    char    id [ 10 ];
    char    name [ 26 ];
    int     gradePts ;
} STUDENT ;

STUDENT aStudent ;
```

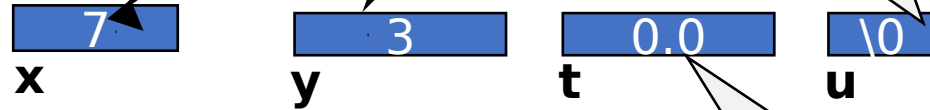
구조체 변수의 초기화

```
typedef struct
{
    int x;
    int y;
    float t;
    char u;
} SAMPLE;
```

SAMPLE sam 1 = { 2, 5, 3.2, 'A' } ;



SAMPLE sam 2 = { 7, 3 }



null ('\0')
로
자동초기화

float 0.0
으로
자동초기화

구조체 변수의 초기화

```
/* 각 구조체의 변수는 3개의 멤버를 가짐 */  
struct student {  
    int number;  
    char name [20] ;  
    int score;  
} part1 = {1, "handong", 95},  
  part2 = {2, "university", 80};
```

part1

1	handong	95
number	name	score

part2

2	university	80
number	name	score

구조체 변수 선언 비교

- 구조체의 변수선언

```
[Ex] struct part {  
    int number;  
    char name [20] ;  
    int on_hand;  
} part1, part2;  
struct part part3, part4;
```

part1, part2, part3, part4
모두 사용가능

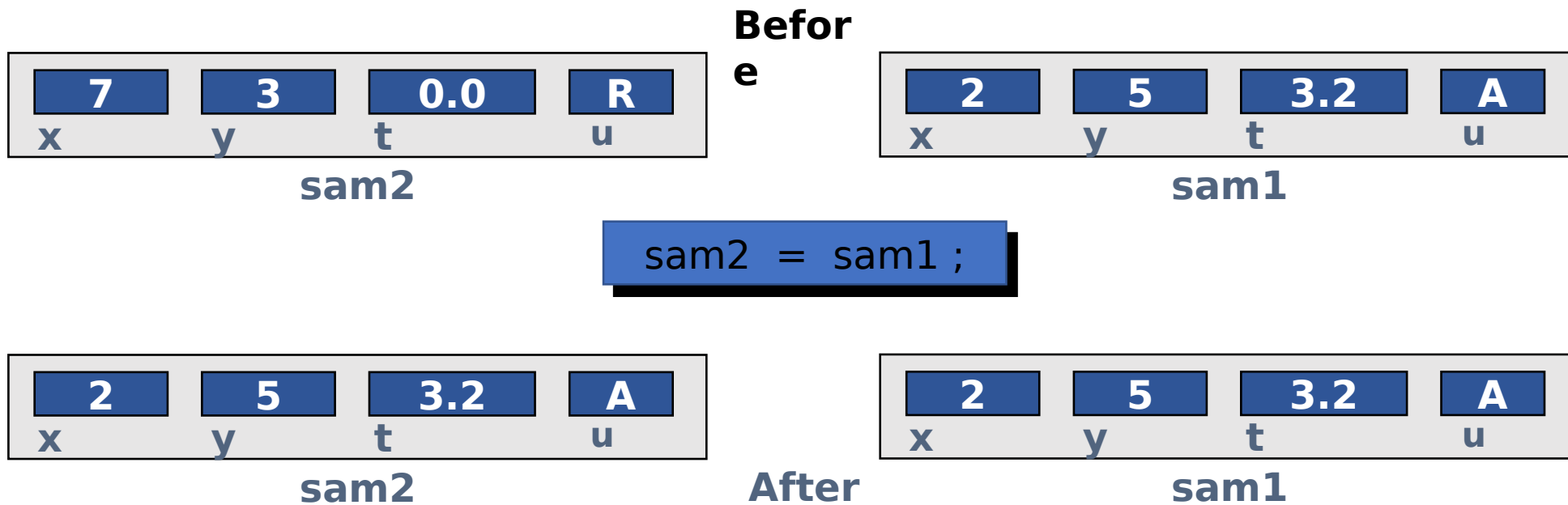
```
[Ex] struct {  
    int number;  
    char name[20];  
    int on_hand;  
} part1;
```

```
struct {  
    int number;  
    char name[20];  
    int on_hand;  
} part2 ;
```

Part1, part2
모두 사용 불가
구조체 tag명이 없고 별칭만 있음
선언은 가능하나 사용불가
typedef를 추가하여 사용가능

구조체 변수와 대입 연산

- 구조체의 복사
 - 구조체 변수에 대입연산자(=)사용 가능



구조체 초기화 예시1

```
#include <stdio.h>

struct person{
    char name[20];
    char phnum[20];
    int age;
};

int main(){
    struct person man, woman={"홍연", "010-1111-2222", 28};

    man=woman;
    printf("man:%s %s %d\n",man.name, man.phnum, man.age);
    printf("woman:%s %s %d\n",woman.name, woman.phnum, woman.age);
    man.name[0]='h';
    man.name[1]='s';
    man.name[2]='t';
    man.name[3]='a';
    man.name[4]='r';
    printf("man:%s %s %d\n",man.name, man.phnum, man.age);
    printf("woman:%s %s %d\n",woman.name, woman.phnum, woman.age);
    return 0;
}
```

```
man:홍연 010-1111-2222, 28 0
woman:홍연 010-1111-2222, 28 0
man:hstar 010-1111-2222, 28 0
woman:홍연 010-1111-2222, 28 0
```

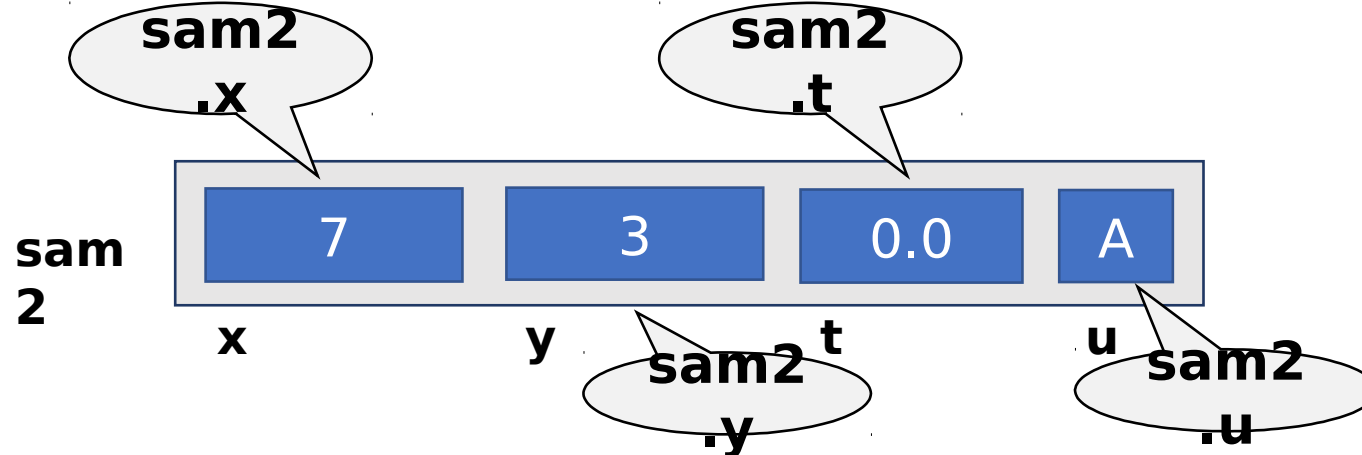
구조체의 멤버 접근 연산자 ‘.’

- 구조체 선언

```
struct sam
{
    int x ;
    int y ;
    float t ;
    char u;
} sam2 ;
```

각 필드는 구조체
요소 또는 멤버라고
부른다

- 구조체 멤버에 접근하기 위해서 ‘.’ 연산자를 사용

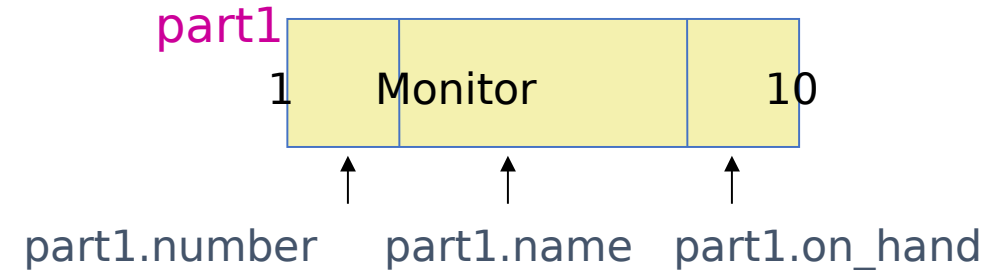


구조체의 멤버 접근 연산자 ‘.’

- 구조체 멤버 접근

```
struct {  
    int number;  
    char name[20];  
    int on_hand;  
} part1={1, "Monitor", 10};  
printf ("Part number : %d\n", part1.number);  
printf ("Part name  : %s\n", part1.name);  
printf ("Quantity on hand : %d\n", part1.on_hand);
```

Part number : 1
Part name : Monitor
Quantity on hand : 10



구조체의 멤버 접근 연산자 ‘.’

```
struct {  
    int number;  
    char name[20];  
    int on_hand;  
} part1;  
  
scanf ("%d", &part1.on_hand);    /* scanf() 데이터 입력*/  
scanf ("%s", part1.name);  
Part1.number = 258;              /* 대입연산자 */  
Part1.on_hand++;                 /* 증감연산자 */
```

구조체의 멤버 접근 연산자 ‘.’

‘.’ 연산자의 우선순위
가장 높은 우선순위를 가지고 있음
왼쪽에서 오른쪽 결합성

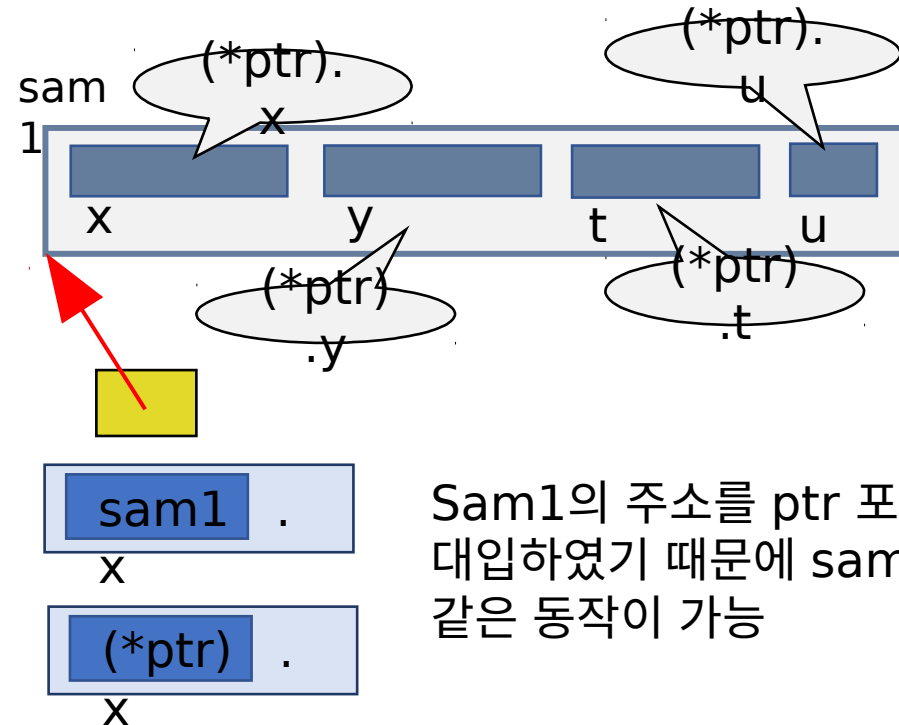
```
typedef struct {  
    int total;  
    char name[10];  
} DATA;  
DATA sales;
```

```
sales.total++;           /* (sales.total)++으로 해석 */  
fun(sales.total);        /* sales.total을 fun함수로 전달 */  
fun1(sales.name[3]);     /* (sales.name)[3]에 있는 값을 fun1함수로 전달 */  
++sales.total;          /* ++(sales.total)으로 해석 */  
fun2(&sales.total);      /* &(sales.total)의 주소값을 fun2로 전달 */
```

구조체와 pointer

- 구조체 포인터 선언

```
typedef struct
{
    int    x ;
    int    y ;
    float  t ;
    char   u ;
} SAMPLE ;
...
SAMPLE    sam1 ;
SAMPLE* ptr ;
...
ptr = &sam1 ;
...
```



Sam1의 주소를 ptr 포인터에
대입하였기 때문에 sam1과
같은 동작이 가능

구조체와 pointer

- 포인터 예제

```
struct complex {  
    double re;  
    double im;  
};  
  
int main(){  
    struct complex c1, c2, *a=&c1, *b=&c2;    /*구조체 변수 c1의 주소를 a에 복사 */  
                                              /*구조체 변수 c2의 주소를 b에 복사 */  
  
    a->re = b->re + 2 ;    /* c1.re = c2.re + 2;와 같은 의미*/  
    b->im = a->im - 3;    /* c2.im = c1.im - 3;와 같은 의미 */  
  
    printf ("value ; %f\n ", a->im);  
}
```


구조체와 pointer

```
#include <stdio.h>

struct shape {
    int x, y ;
    char name[10] ;
};

int main(){
    struct shape s, *p = &s;
    scanf ("%d %d %s", &p->x , &p->y, p->name );
    /*연산자 '->'는 '&'보다 우선순위가 높음
       &p->x == &(p->x)*/

    s.x = p->x * 2; /* p->x = s.x *2;와 같은 의
    미 */
    p->y = s.y % 5;
    printf ("%d %d %s\n", p->x , p->y, p->name ) ;
    return 0;
}
```

```
1 2 hustar
2 2 hustar
```

같은 결과

*p.y 는 *(p.y)로 해석됨 따라서 오류

[Ex]

```
struct shape s, *p = &s;
scanf ("%d %d %s", &(*p).x , &(*p).y, (*p).name ) ;

/* 연산자 '->'는 '&'보다 높은 우선 순위를 가짐.
   &p->x == &(*p).x*/

s.x = (*p).x * 2;
(*p).y = s.y % 5;
printf ("%d %d %s\n", (*p).x , (*p).y, (*p).name ) ;
```

구조체와 pointer 예제

```
#include <stdio.h>

struct card {
    char *face;    // 포인터 face
    char *suit;    // 포인터 suit
};

int main( ){
    struct card aCard;        // 구조체 변수 aCard 정의
    struct card *cardPtr;     // struct card 포인터변수 정의

    // aCard에 string 대입
    aCard.face = "Ace";
    aCard.suit = "Spades";

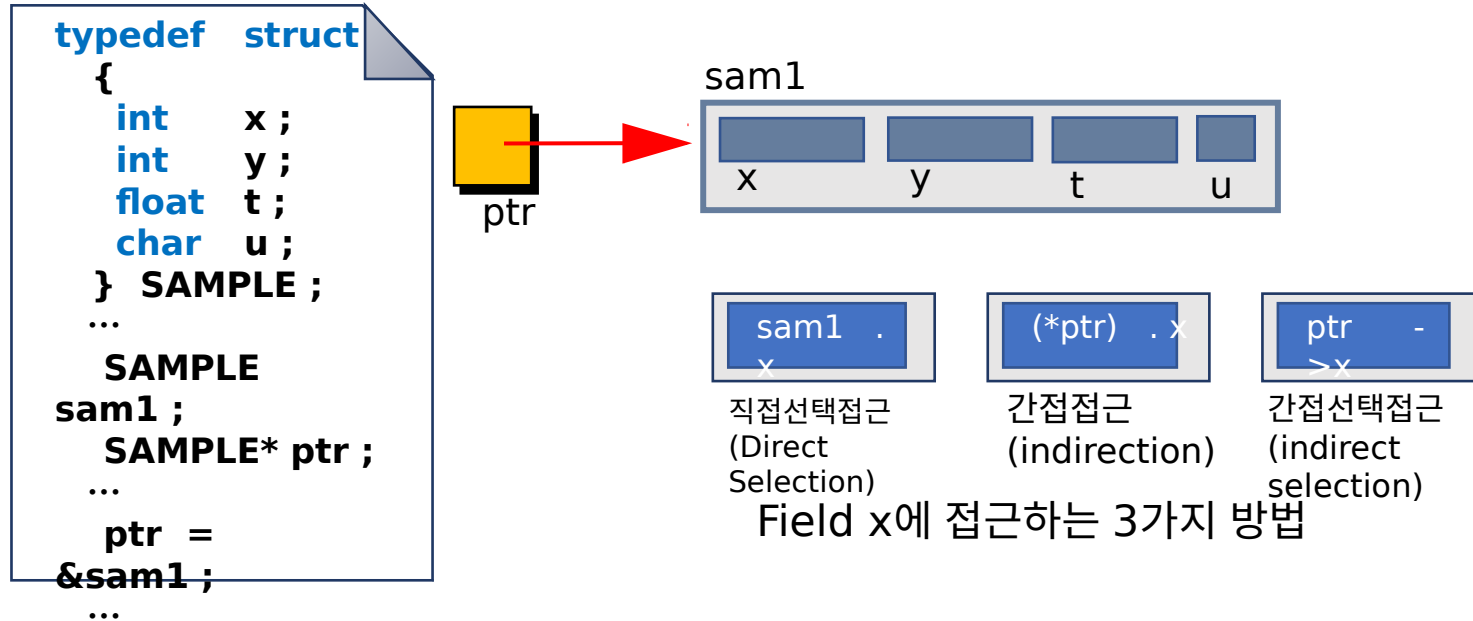
    cardPtr = &aCard;         //cardPtr에 aCard 주소 할당

    printf("%s%s\n%s%s\n%s%s\n", aCard.face, " of ", aCard.suit, cardPtr->face, " of ",
           cardPtr->suit, ( *cardPtr ).face, " of ", ( *cardPtr ).suit );
}
```

```
Ace of Spades
Ace of Spades
Ace of Spades
```

구조체 연산자

- 간접연산자(indirect selection operator)



구조체 실습1

- 제품명(pro_name[10]), 제품가격(pro_value)를 가지는 구조체를 이용하는 프로그램 작성
 - 제품명은: TV, Radio(예시)
 - 제품가격: 30000, 10000(예시)
 - 최종 합계 가격을 출력
- 1개의 제품을 user 입력
- 2번째 제품은 프로그램 코드에서 입력(반드시 첫 번째 상품을 입력 이 후 두 번째 제품의 이름과 가격을 입력할 것)

제품명과 가격을 입력 : TV 30000

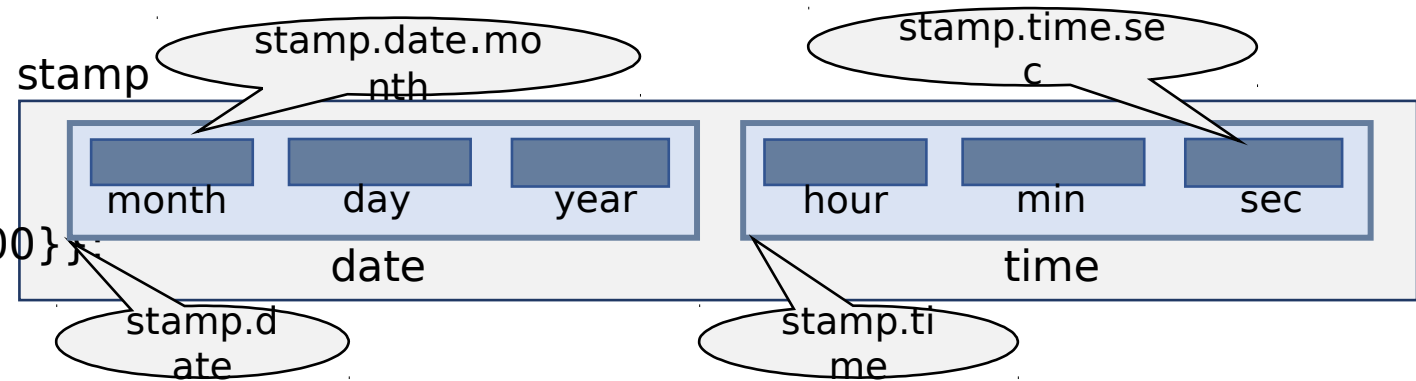
제품명	가격
TV	30000
Radio	15000
sum	45000

중첩구조체

- 구조체 안에 구조체가 멤버로 있는 구조

```
typedef struct{
    int month, day, year;
} DATE;
typedef struct{
    int hour, min, sec;
} TIME;
typedef struct{
    DATE date; TIME time;
} STAMP;
/* 변수 선언 및 초기화*/
STAMP stamp={{01,01,07},{23,45,00}}
```

```
typedef struct{
    STAMP start;
    STAMP end;
}JOB;
JOB job; /* 변수 선언*/
```



중첩구조체

- 중첩구조체 선언 방법
 - 하나의 구조체 안에서 중첩하여 선언
 - 두 개의 구조체를 선언하여 중첩시키는 것
 - 구조체를 중첩할 때 멤버로 포함되는 구조체는 반드시 선언되어 있어야 멤버로 사용가능

```
struct person {  
    struct {  
        char first[10],  
        char last[10];  
    } name;  
    char address[30];  
    int age;
```

} 하나의 구조체 안에
중첩하여 선언

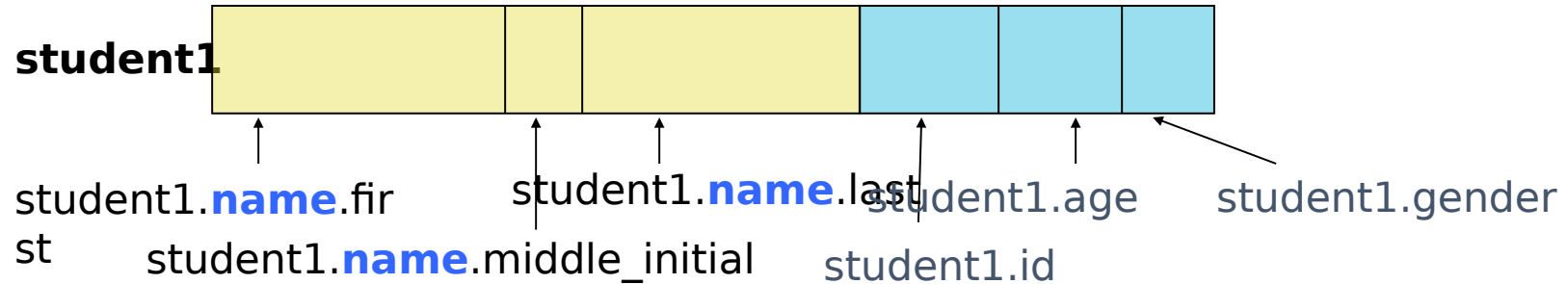
```
struct nam{  
    char first[10];  
    char last[10];  
};  
  
struct person {  
    struct nam name;  
  
    char address[30];  
    int age;  
}; //명확하고 간단  
두 개의 구조체 선언
```

중첩구조체

- 중첩구조체의 참조

```
struct person_name {  
    char first[10];  
    char middle_initial;  
    char last[10];  
};
```

```
struct student {  
    struct person_name name ;  
    int id, age;  
    char gender;  
} student1, student2;
```



중첩구조체

- 중첩구조체 참조
 - ‘.’ 연산자를 이용

```
strcpy ( student1.name.first, "Fred" );  
strcpy ( student1.name.last, "Gordon" );
```

```
display_name ( student1.name);    /* 하나의 인수 전달*/
```

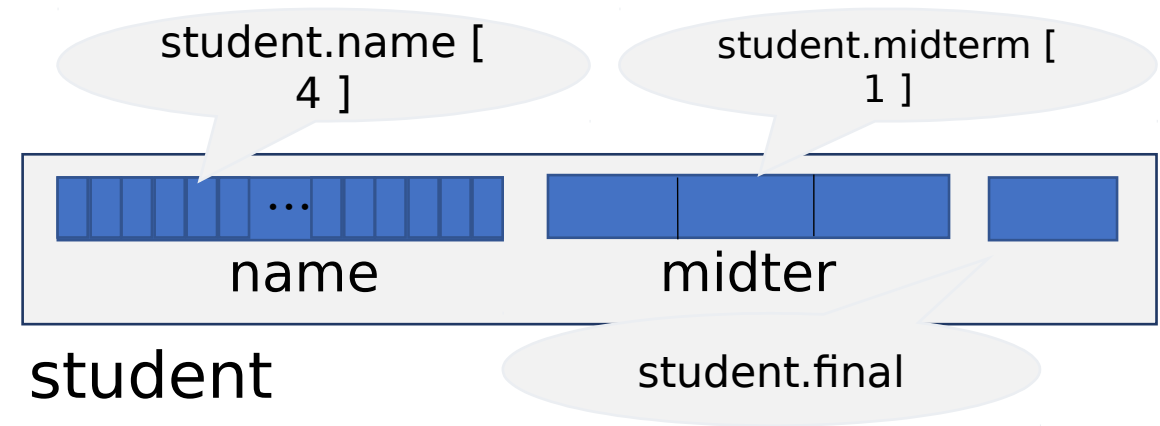
```
struct person_name new_name={"Young",'s', "Kim"} ;  
student1.name = new_name ;      /* 대입 연산자 사용가능 */
```


구조체에서 배열 멤버 사용

- 구조체에는 배열을 멤버로 사용할 수 있음

```
// 전역 선언 공간
typedef struct{
    char name[26];
    int midterm[3];
    int final;
} STUDENT;

STUDENT student ={"Hong Kil-Dong",{90,75,80},92};
int *p, total;
:
p=student.midterm;
total= *p+*(p+1)+*(p+2);
/* Same as total=p[0]+p[1]+p[2]; */
```



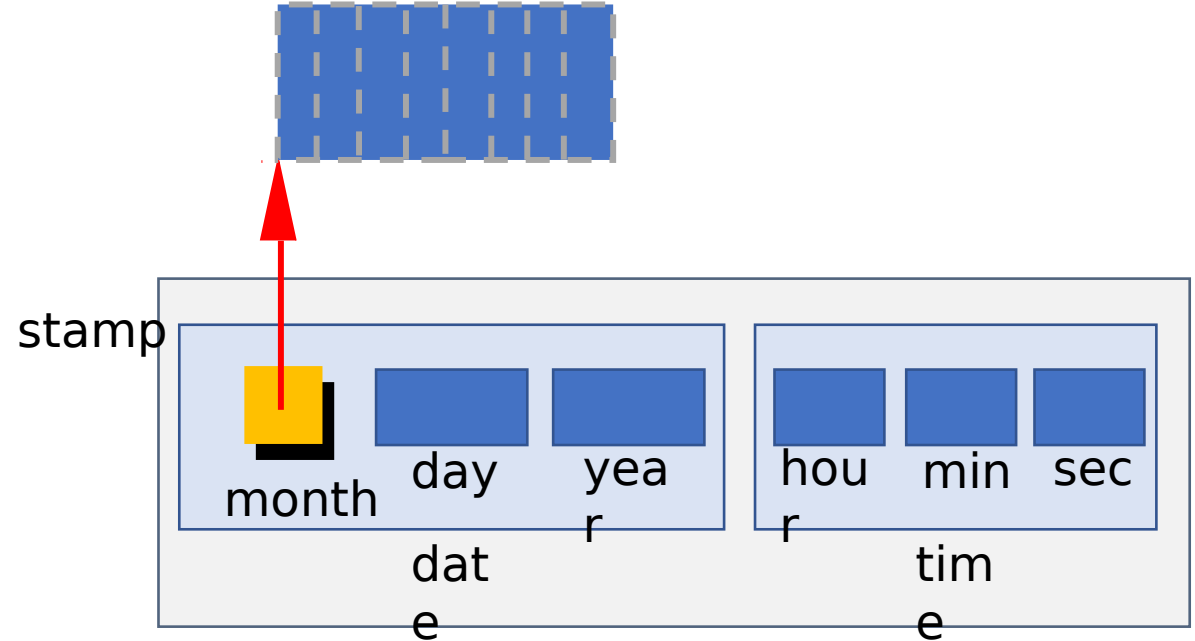
구조체에서 포인터 멤버

- 구조체에서 포인터를 멤버로 포함시킬 수 있음

```
typedef struct{  
    int hour, min, sec;  
} CLOCK;
```

```
typedef struct{  
    char * month;  
    int day, year;  
} DAY;
```

```
struct stp{  
    DAY date;  
    CLOCK time;  
}stamp;
```



구조체에서 포인터 멤버

- 구조체에서 포인터를 멤버로 포함시킬 수 있음

```
typedef struct{
    char *name;          /* 26 char 배열을 대신하여 4 byte pointer */
    int midterm[3];
    int final;
} STUDENT;
STUDENT student = {"Hong Kil-Dong", {90, 75, 80}, 92};

gets(student.name);     /* error: name을 위한 저장공간이 없음*/
```

Name을 위한 메모리 할당이 필요.
student.name=malloc(26);
gets(student.name);

- 구조체를 배열로 선언

STUDENT stuary[50];

stuary[49]

•
•
•

구조체 배열

- 초기화

```
STUDENT stuary[] = {{“kim”, 67, 89, 90, 70},  
                    {“choi”, 66, 77, 88, 99},  
                    {“Lee”, 50, 78, 98, 80},  
                    {“hong”, 76, 68, 79, 80}} ;
```

- 개별 요소에 접근하기 위해서는 index를 이용

[Ex]

```
STUDENT *pstu;
```

```
pstu=stuary;
```

```
stuary[1] = stuary[0]; // stuary[0]의 모든 멤버를 stuary[1]로  
복사
```

```
*(pstu+1) = *pstu; // stuary[1]=stuary[0] 와 같은 의미
```

구조체 배열 예시

```
#include <stdio.h>

struct person
{
    char name[20];
    char phoneNum[20];
    int age;
};

int main(void)
{
    struct person arr[3]={
        {"이승기", "010-1212-0001", 21}, // 첫 번째 요소의 초기화
        {"정지영", "010-1313-0002", 22}, // 두 번째 요소의 초기화
        {"한지수", "010-1717-0003", 19}  // 세 번째 요소의 초기화
    };

    int i;
    for(i=0; i<3; i++)
        printf("%s %s %d \n", arr[i].name, arr[i].phoneNum, arr[i].age);

    return 0;
}
```

이승기	010-1212-0001	21
정지영	010-1313-0002	22
한지수	010-1717-0003	19

구조체와 함수

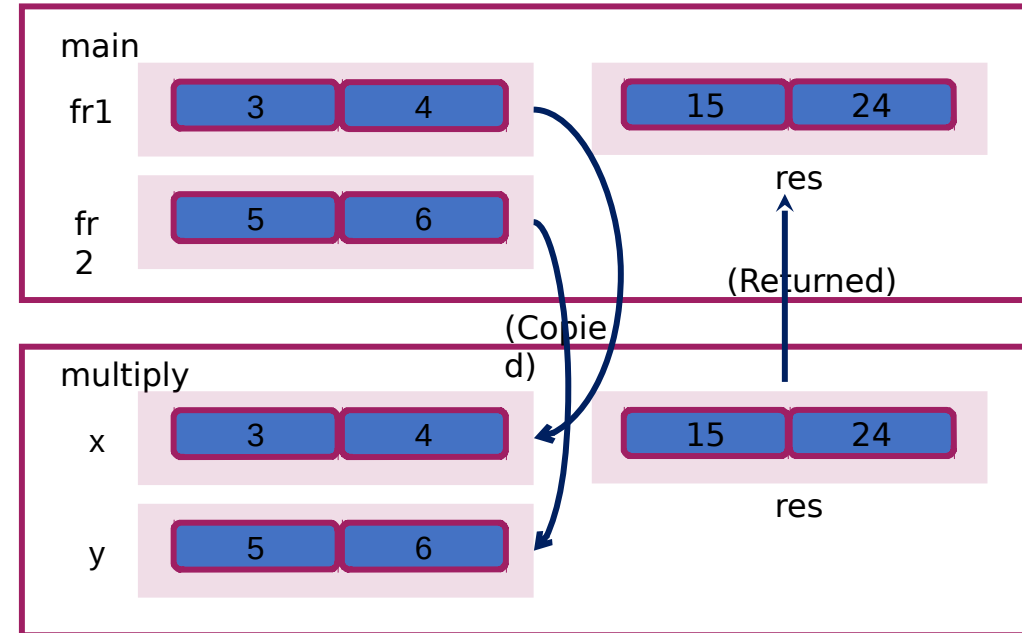
- Call by value(pass by value)
 - 구조체의 멤버를 함수로 전달
 - 인수는 일반 변수처럼 독립적으로 취급
 - 복사하여 사본을 함수에 전달
 - 값이 복사되어 전달되기 때문에 원본데이터를 변형 불가
- Call by reference(pass by address)
 - 구조체의 주소를 전달
 - 함수는 간접적으로 전체 구조체에 접근 가능하며 작업을 할 수 있음

구조체와 함수

- 전체 구조체를
- 복사를 전달

```
FRACTION multiply(FRACTION, FRACTION);  
FRACTION fr1={3, 4}, fr2={5, 6}, res;  
  
res =multiply(fr1, fr2);
```

```
FRACTION multiply (FRACTION x, FRACTION y)  
{  
    FRACTION res;  
    res.numerator = x.numerator* y.numerator;  
    res.denominator=x.denominator*y.denominator;  
    return res;  
}
```

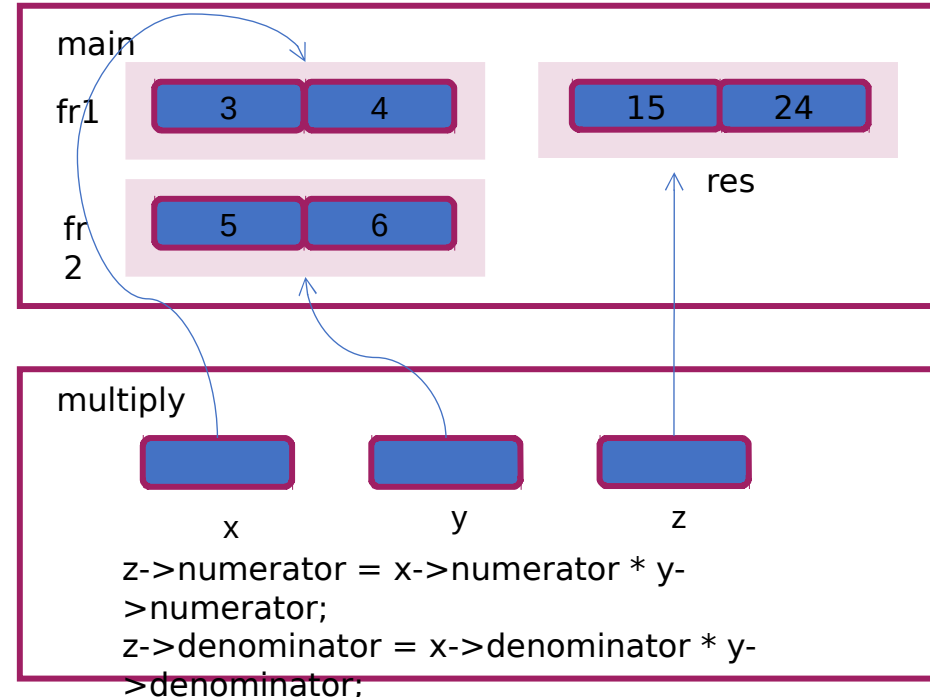


구조체와 함수

- 포인터를 통하여 구조체의 주소를 전달

```
void multiply(FRACTION*, FRACTION*, FRACTION*);  
FRACTION fr1={3, 4}, fr2={5, 6}, res;  
  
multiply(&fr1, &fr2, &res);
```

```
void multiply (FRACTION *x, FRACTION *y, FRACTION  
*z)  
{  
    z->numerator = x->numerator * y->numerator;  
    z->denominator = x->denominator * y-  
>denominator;  
}
```



구조체와 함수 비교

전체 구조체를 복사하여 전달하는 예제

```
struct card {  
    int    pips;  
    char  suit;  
} c1 = {5, 'd'};
```

포인터를 통한 구조체의 전달 예제

[Ex] 전체 구조체의 전달

```
:  
    assign_values(c1);  
    printf("%d %c", c1.pips, c1.suit);  
:  
void assign_values(struct card c) {  
    c.pips = 1;  
    c.suit = 'c';  
}
```

5 d

[Ex] 포인터를 통한 구조체의 전달

```
:  
    assign_values(&c1);  
    printf("%d %c", c1.pips, c1.suit);  
:  
void assign_values(struct card *c) {  
    c->pips = 1;  
    c->suit = 'c';  
}
```

1 c

구조체와 함수 예시

- 포인터를 통한 구조체의 전달

```
#include <stdio.h>

struct invent {
    char name[20];
    int number;
    double price;
};

void input(struct invent*);

void input(struct invent*ptr){
    int i;

    printf("INPUT\n\n");

    for(i=0; i<3; i++,ptr++)
        scanf("%s %d %lf", ptr->name, &ptr->number, &ptr->price);
}
```

```
int main() {
    struct invent product[3], *ptr;

    input(product);
    printf("\nOUTPUT\n\n");
    ptr = product;
    while(ptr < product + 3) {
        printf("%-20s %5d %10.2f\n", ptr->name, ptr->number, ptr->price);
        ptr++;
    }
}
```

INPUT

```
name 123 12300
kkk 321 32100
good 212 449
```

OUTPUT

name	123	12300.00
kkk	321	32100.00
good	212	449.00

구조체 배열 예제2, element printf

- 구조체 배열 아이템 출력

```
struct invent {  
    char name[20];  
    int  number;  
    double price;  
};  
void output(struct invent *);  
  
int main() {  
    struct invent product[100];  
    .....  
    printf("\nOUTPUT\n\n");  
    output(product)  
}
```

// fill data

```
void output(struct invent *p){  
  
    for( ; p < p + 100; p++)  
        printf("%-20s %5d %10.2f\n", p->name, p->number, p->price);  
  
}
```

구조체 예제3

• 카드 이름 출력

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

```
#define CARDS 52  
#define FACES 13
```

```
// 카드의 구조체 정의
```

```
struct card {  
    const char *face;    // define pointer face  
    const char *suit;    // define pointer suit  
};
```

```
// new type name for struct card
```

```
typedef struct card Card;
```

```
void fillDeck(Card *wDeck, const char * wFace[], const char * wSuit[]);
```

```
void deal( const Card * wDeck );
```

```
int main( void ){  
    Card deck[ CARDS ];    // define array of Cards  
  
    // initialize array of pointers  
    const char *face[] = { "Ace", "Deuce", "Three", "Four", "Five",  
        "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King"};  
  
    // initialize array of pointers  
    const char *suit[] = { "Hearts", "Diamonds", "Clubs", "Spades"};  
  
    fillDeck( deck, face, suit );    // load the deck with Cards  
    deal( deck );    // deal all 52 Cards  
}
```

구조체 예제3

```
void fillDeck( Card * wDeck, const char * wFace[], const char * wSuit
[] ){
    int i, j, k;

    for ( i = 0; i < CARDS; ++i ) {
        j=i%FACES;  k=i/FACES;
        wDeck[i].face = wFace[ j ];
        wDeck[i].suit = wSuit[ k ];
    }
}

void deal( const Card * wDeck ){
    int i;

    for ( i = 0; i < CARDS; ++i ) {
        printf( "%5s of %-7s", wDeck[ i ].face, wDeck[ i ].suit);
        if (!((i+1)%4)) putchar('\n'); }
}
```

```
Ace of Hearts Deuce of Hearts Three of Hearts Four of Hearts
Five of Hearts Six of Hearts Seven of Hearts Eight of Hearts
Nine of Hearts Ten of Hearts Jack of Hearts Queen of Hearts
King of Hearts Ace of Diamonds Deuce of Diamonds Three of Diamonds
Four of Diamonds Five of Diamonds Six of Diamonds Seven of Diamonds
Eight of Diamonds Nine of Diamonds Ten of Diamonds Jack of Diamonds
Queen of Diamonds King of Diamonds Ace of Clubs Deuce of Clubs
Three of Clubs Four of Clubs Five of Clubs Six of Clubs
Seven of Clubs Eight of Clubs Nine of Clubs Ten of Clubs
Jack of Clubs Queen of Clubs King of Clubs Ace of Spades
Deuce of Spades Three of Spades Four of Spades Five of Spades
Six of Spades Seven of Spades Eight of Spades Nine of Spades
Ten of Spades Jack of Spades Queen of Spades King of Spades
```