

CHAPTER 6

구조체와 포인터 그리고  
함수 공용체와 열거형

# 학습 목표

- 구조체의 정의와 typedef 선언



# 구조체의 정의와 typedef 선언

# Typedef 선언

자료형의 이름 *int*에 *INT*라는 이름을 추가로 붙여줍니다.



위의 *typedef* 선언으로 인해서!!!

*int num;* 과 동일한 선언

*int \* ptr;* 과 동일한 선언

새로 부여된 이름	대상 자료형
INT	int
PTR_INT	int *
UINT	unsigned int
PTR_UINT	unsigned int *
UCHAR	unsigned char
PTR_UCHAR	unsigned char *

정의되는 이름들

실행결과

120, 190, Z

```
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(void)
{
    INT num1 = 120;           // int num1 = 120;
    PTR_INT pnum1 = &num1;    // int * pnum1 = &num1;

    UINT num2 = 190;          // unsigned int num2 = 190;
    PTR_UINT pnum2 = &num2;    // unsigned int * pnum2 = &num2;

    UCHAR ch = 'Z';           // unsigned char ch = 'Z';
    PTR_UCHAR pch = &ch;      // unsigned char * pch = &ch;

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

# 구조체 정의와 typedef 선언

```
struct point
{
    int xpos;
    int ypos;
};

typedef struct point Point;
```

구조체 *point* 정의 후

*struct point*에 *Point*라는 이름을 부여하기 위한 *typedef* 선언 추가!



합친 형태

```
typedef struct point
{
    int xpos;
    int ypos;
} Point;
```

구조체 *point*의 정의와

*Point*에 대한 *typedef* 선언을 한데 묶은 형태

# 구조체 정의와 typedef 선언 관련 예제

```
struct point
{
    int xpos;
    int ypos;
};
```

typedef struct point Point; 구조체 point의 정의와 typedef 선언

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

구조체 person의 정의와

Person이라는 이름의 typedef 선언을 하나로!

```
int main(void)
{
    Point pos={10, 20};
    Person man={"이승기", "010-1212-0001", 21};
    printf("%d %d \n", pos.xpos, pos.ypos);
    printf("%s %s %d \n", man.name, man.phoneNum, man.age);
    return 0;
}
```

실행결과

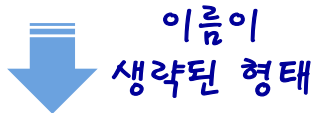
10 20

이승기 010-1212-0001 21

# 구조체의 이름 생략

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

*typedef* 선언으로 인해서 새로운 이름 *Person*이 정의되었으니,  
구조체의 이름 *persons*은 큰 의미가 없다.



```
typedef struct 
{
    char name[20];
    char phoneNum[20];
    int age;
} Person;
```

따라서 이렇듯 구조체의 이름을 생략하는 것도 가능하다.

# 함수로의 구조체 변수 전달과 반환



# 함수의 인자로 전달되고 return 문에 의해 반환되는 구조체 변수 1

```
typedef struct point
{
    int xpos;
    int ypos;
} Point;

void ShowPosition(Point pos)
{
    printf("[%d, %d] \n", pos.xpos, pos.ypos);
}

Point GetCurrentPosition(void)
{
    Point cen;
    printf("Input current pos: ");
    scanf("%d %d", &cen.xpos, &cen.ypos);
    return cen; 구조체 변수 cen이 통째로 반환된다.
}

int main(void)
{
    Point curPos=GetCurrentPosition();
    ShowPosition(curPos);
    return 0; ShowPosition 함수의 매개변수에 curPos에 저장된 값이 통째로 복사된다.
}
```

실행결과

Input current pos: 2 4  
[2, 4]

# 배열까지도 통째로 복사

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

구조체의 멤버로 배열이 선언된 경우  
구조체 변수를 인자로 전달하거나 반환 시  
배열까지도 통째로 복사가 이뤄진다.

실행결과

```
name? Jung
phone? 010-12XX-34XX
age? 22
name: Jung
phone: 010-12XX-34XX
age: 22
```

```
void ShowPersonInfo(Person man)
{
    printf("name: %s \n", man.name);
    printf("phone: %s \n", man.phoneNum);
    printf("age: %d \n", man.age);
}

Person ReadPersonInfo(void)
{
    Person man;
    printf("name? "); scanf("%s", man.name);
    printf("phone? "); scanf("%s", man.phoneNum);
    printf("age? "); scanf("%d", &man.age);
    return man;
}

int main(void)
{
    Person man=ReadPersonInfo();
    ShowPersonInfo(man);
    return 0;
}
```

# 구조체 기반의 Call-by-reference

```
typedef struct point
{
    int xpos;
    int ypos;
} Point;

void OrgSymTrans(Point * ptr)    // 원점대칭
{
    ptr->xpos = (ptr->xpos) * -1;
    ptr->ypos = (ptr->ypos) * -1;
}

void ShowPosition(Point pos)
{
    printf("[%d, %d] \n", pos.xpos, pos.ypos);
}

int main(void)
{
    Point pos={7, -5};
    OrgSymTrans(&pos);    // pos의 값을 원점 대칭이동시킨다.
    ShowPosition(pos);
    OrgSymTrans(&pos);    // pos의 값을 원점 대칭이동시킨다.
    ShowPosition(pos);
    return 0;
}
```

구조체 변수 대상의 *Call-by-reference*는  
일반변수의 *Call-by-reference*와 동일하다.

실행결과

[-7, 5]

[7, -5]

# 구조체 변수를 대상으로 가능한 연산1

```
typedef struct point
{
    int xpos;
    int ypos;
} Point;
```

```
int main(void)
{
```

```
    Point pos1={1, 2};
```

```
    Point pos2;
```

```
    pos2=pos1; // pos1의 멤버 대 pos2의 멤버간 복사가 진행됨
```

```
    printf("크기: %d \n", sizeof(pos1)); // pos1의 전체 크기 반환
```

```
    printf("[%d, %d] \n", pos1.xpos, pos1.ypos);
```

```
    printf("크기: %d \n", sizeof(pos2)); // pos2의 전체 크기 반환
```

```
    printf("[%d, %d] \n", pos2.xpos, pos2.ypos);
```

```
    return 0;
```

```
}
```

구조체 변수간 대입 연산의 결과로 멤버 대 멤버 복사가 이뤄진다는 사실을  
확인하자!

실행결과

크기: 8

[1, 2]

크기: 8

[1, 2]

# 구조체 변수를 대상으로 가능한 연산2

```
typedef struct point
{
    int xpos;
    int ypos;
} Point;
```

구조체 변수를 대상으로는 덧셈 및 뺄셈 연산이 불가능하다.

따라서 필요하다면 덧셈함수와 뺄셈함수를 정의해야 한다.

실행결과

```
[7, 15]
[3, -3]
```

```
Point AddPoint(Point pos1, Point pos2)
{
    Point pos={pos1.xpos+pos2.xpos, pos1.ypos+pos2.ypos};
    return pos;
}

Point MinPoint(Point pos1, Point pos2)
{
    Point pos={pos1.xpos-pos2.xpos, pos1.ypos-pos2.ypos};
    return pos;
}

int main(void)
{
    Point pos1={5, 6};
    Point pos2={2, 9};
    Point result;

    result=AddPoint(pos1, pos2);
    printf("[%d, %d] \n", result.xpos, result.ypos);
    result=MinPoint(pos1, pos2);
    printf("[%d, %d] \n", result.xpos, result.ypos);
    return 0;
}
```

구조체 Point의 덧셈 함수

구조체 Point의 뺄셈 함수



# 구조체의 유용함에 대한 논의와 중첩 구조체

# 구조체를 정의하는 이유

- ▶ 연관 있는 데이터를 하나로 묶을 수 있는 자료형을 정의할 수 있다. 구조체의 정의 이유!
- ▶ 연관 있는 데이터를 묶으면 데이터의 표현 및 관리가 용이해진다.
- ▶ 데이터의 표현 및 관리가 용이해지면 그만큼 합리적인 코드를 작성할 수 있다.

```
typedef struct student
{
    char name[20];        // 학생 이름
    char stdnum[20];       // 학생 고유번호
    char school[20];       // 학교 이름
    char major[20];        // 선택 전공
    int year;              // 학년
} Student;

void ShowStudentInfo(Student * sptr)
{
    printf("학생 이름: %s \n", sptr->name);
    printf("학생 고유번호: %s \n", sptr->stdnum);
    printf("학교 이름: %s \n", sptr->school);
    printf("선택 전공: %s \n", sptr->major);
    printf("학년: %d \n", sptr->year);
}
```

인자 전달 시 용이하다.

```
int main(void)
{
    Student arr[7];
    int i;

    for(i=0; i<7; i++)
    {
        printf("이름: "); scanf("%s", arr[i].name);
        printf("번호: "); scanf("%s", arr[i].stdnum);
        printf("학교: "); scanf("%s", arr[i].school);
        printf("전공: "); scanf("%s", arr[i].major);
        printf("학년: "); scanf("%d", &arr[i].year);
    }

    for(i=0; i<7; i++)
        ShowStudentInfo(&arr[i]);

    return 0;
}
```

하나의 배열 선언으로 종류가 다른 데이터들을 한데 저장할 수 있다.

# 중첩된 구조체의 정의와 변수의 선언

```
typedef struct point
{
    int xpos;
    int ypos;
} Point;
```

```
typedef struct circle
{
    Point cen;
    double rad;
} Circle;
```

```
void ShowCircleInfo(Circle * cptr)
{
    printf("[%d, %d] \n", (cptr->cen).xpos, (cptr->cen).ypos);
    printf("radius: %g \n\n", cptr->rad);
}

int main(void)
{
    Circle c1={{1, 2}, 3.5};
    Circle c2={{2, 4}, 3.9};
    ShowCircleInfo(&c1);
    ShowCircleInfo(&c2);
    return 0;
}
```

앞서 정의한 구조체는 이후에 새로운 구조체를 선언하는데 있어서  
기본 자료형의 이름과 마찬가지로 사용이 될 수 있다.

실행결과


[1, 2]  
radius: 3.5

[2, 4]  
radius: 3.9




# 공용체(Union Type)의 정의와 의미

# 구조체 vs 공용체 : 선언방식의 차이



```
typedef struct sbbox
{
    int mem1;
    int mem2;
    double mem3;
} SBox;
```



```
typedef union ubox
{
    int mem1;
    int mem2;
    double mem3;
} UBox;
```

정의 방법에 있어서의 차이는 키워드 *struct*를 쓰느냐, 아니면 키워드 *union*을 쓰느냐에 있다!

# 구조체 vs 공용체 : 실행결과를 통한 관찰

공용체 변수를 이루는 멤버의 시작 주소 값이

모두 동일함을 관찰하고 공용체 변수의 크기 값을 관찰한다!

실행결과

```
002CFC28 002CFC2C 002CFC30
002CFC18 002CFC18 002CFC18
16 8
```

```
int main(void)
{
    SBox sbx;
    UBox ubx;
    printf("%p %p %p \n", &sbx.mem1, &sbx.mem2, &sbx.mem3);
    printf("%p %p %p \n", &ubx.mem1, &ubx.mem2, &ubx.mem3);
    printf("%d %d \n", sizeof(SBox), sizeof(UBox));
    return 0;
}
```

```
typedef struct sbox
{
    int mem1;
    int mem2;
    double mem3;
} SBox;

typedef union ubox
{
    int mem1;
    int mem2;
    double mem3;
} UBox;
```

# 구조체 vs 공용체 : 메모리적 차이

```
typedef union ubox    // 공용체 ubox의 정의
{
    int mem1;
    int mem2;
    double mem3;
} UBox;

int main(void)
{
    UBox ubx;    // 8바이트 메모리 할당
    ubx.mem1=20;
    printf("%d \n", ubx.mem2);

    mem1에 저장된 데이터를 덮어쓴다.
    ubx.mem3=7.15;
    printf("%d \n", ubx.mem1);
    printf("%d \n", ubx.mem2);
    printf("%g \n", ubx.mem3);
    return 0;
}
```

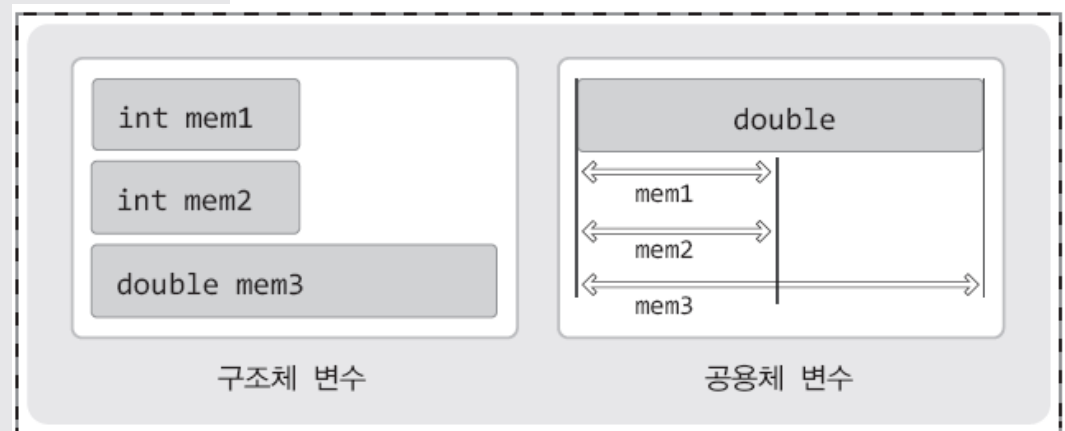
20

-1717986918

-1717986918

7.15

실행결과



# 공용체의 유용함1 : 문제의 제시

- 민선: 수진아! 교수님이 과제를 내 주셨어
- 수진: 뭔데?
- 민선: 프로그램 사용자로 부터 int형 정수 하나를 입력 받으래
- 수진: 그래서?
- 민선: 입력 받은 정수의 상위 2바이트와 하위 2바이트 값을 양의 정수로 출력!
- 수진: 그게 다야?
- 민선: 그 다음엔 상위 1바이트와 하위 1바이트에 저장된 값의 아스키 문자 출력!
- 수진: 그거 공용체를 이용해 보라는 깊은 뜻이 담겨있는 것 같은데?



```
typedef struct dbshort
{
    unsigned short upper;
    unsigned short lower;
} DBShort;

typedef union rdbuf
{
    int iBuf;
    char bBuf[4];
    DBShort sBuf;
} RDBuf;
```

해결책이 되는 공용체의 정의

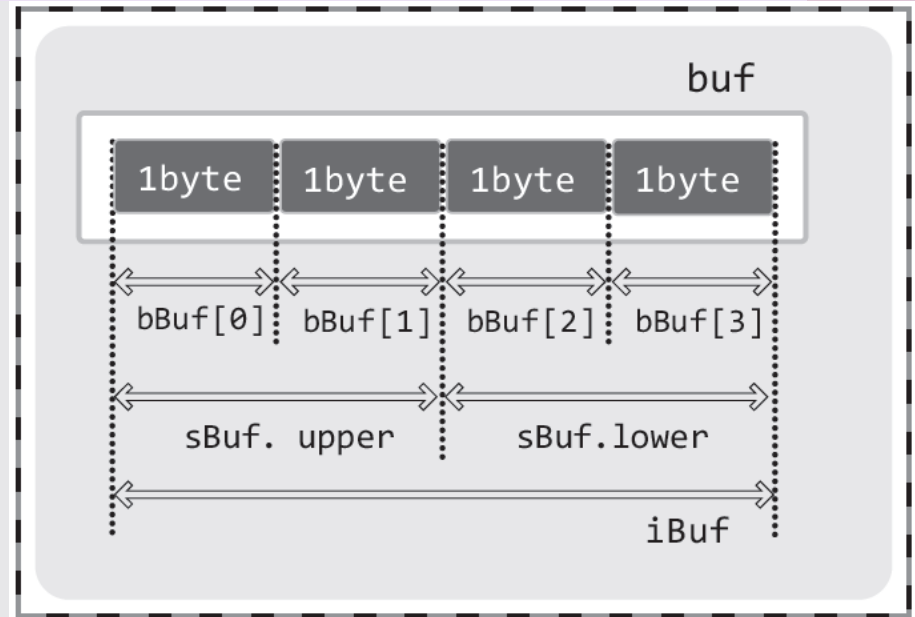
# 공용체의 유용함 2: 문제의 해결

```
typedef struct dbshort
{
    unsigned short upper;
    unsigned short lower;
} DBShort;

typedef union rdbuf
{
    int iBuf;
    char bBuf[4];
    DBShort sBuf;
} RDBuf;

int main(void)
{
    RDBuf buf;
    printf("정수 입력: ");
    scanf("%d", &(buf.iBuf));

    printf("상위 2바이트: %u \n", buf.sBuf.upper);
    printf("하위 2바이트: %u \n", buf.sBuf.lower);
    printf("상위 1바이트 아스키 코드: %c \n", buf.bBuf[0]);
    printf("하위 1바이트 아스키 코드: %c \n", buf.bBuf[3]);
    return 0;
}
```



## 실행결과

```
정수 입력: 1145258561
상위 2바이트: 16961
하위 2바이트: 17475
상위 1바이트 아스키 코드: A
하위 1바이트 아스키 코드: D
```



# 열거형(Enumerated Type)의 정의와 의미

# 열거형의 정의와 변수의 선언

## ▶ 열거형 `syllable`의 정의의 의미

“`syllable`형 변수에 저장이 가능한 정수 값들을 결정하겠다”

## ▶ 열거형 `syllable`의 정의의 예

```
enum syllable    // syllable이라는 이름의 열거형 정의
{
    Do=1, Re=2, Mi=3, Fa=4, So=5, La=6, Ti=7
};
```

*Do*를 정수 1을 의미하는 상수로 정의한다.

그리고 이 값은 `syllable`형 변수에 저장이 가능하다

*Do, Re, Mi, Fa...*

를 열거형 상수라 한다.

## ▶ `syllable`형 변수의 선언

```
enum syllable tone;    // syllable형 변수 tone의 선언
```

구조체 공용체와 마찬가지로 `typedef` 선언을 추가하여 `enum` 선언을 생략할 수 있다.



# 열거형의 정의와 변수선언의 예

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

*typedef 선언이 추가된 열거형의 정의 및 선언*

실행결과

```
도는 하얀 도라지 ♪
레는 둥근 레코드 ♪
미는 파란 미나리 ♪ ♪
파는 예쁜 파랑새 ♪ ♪
술은 작은 술방울 ♪ ♪ ♪
라는 라디오고요~ ♪ ♪ ♪ ♪
시는 졸졸 시냇물 ♪ ♪ ♪ ♪
```

```
void Sound(Syllable sy)
{
    switch(sy)
    {
        case Do:
            puts("도는 하얀 도라지 ♪"); return;
        case Re:
            puts("레는 둥근 레코드 ♪"); return;
        case Mi:
            puts("미는 파란 미나리 ♪ ♪"); return;
        case Fa:
            puts("파는 예쁜 파랑새 ♪ ♪"); return;
        case So:
            puts("술은 작은 술방울 ♪ ♪ ♪"); return;
        case La:
            puts("라는 라디오고요~ ♪ ♪ ♪ ♪"); return;
        case Ti:
            puts("시는 졸졸 시냇물 ♪ ♪ ♪ ♪"); return;
    }
    puts("다 함께 부르세~ 도레미파 솔라시도 솔 도~ 째~");
}

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

*열거형 상수는 선언 이후 어디서건  
쓸 수 있는 상수가 된다.*

# 열거형 상수의 값이 결정되는 방식

```
enum color {RED, BLUE, WHITE, BLACK};
```



동일한 선언

```
enum color {RED=0, BLUE=1, WHITE=2, BLACK=3};
```

열거형 상수의 값은 명시되지 않으면 0부터 시작해서 1씩 증가한다.

```
enum color {RED=3, BLUE, WHITE=6, BLACK};
```



동일한 선언

```
enum color {RED=3, BLUE=4, WHITE=6, BLACK=7};
```

값이 명시되지 않은 상수는 앞에 정의된 상수 값에서 1이 증가한다.

# 열거형의 유용함

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

Syllable이라는 이름의 자료형 안에서 음계에  
관련있는 상수들을 모두 묶어서 정의하였다!

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

변수의 선언이 목적이 아닌 상수의 선언이  
목적인 경우 이렇듯 열거형의 이름과 typedef  
선언을 생략하기도 한다.

열거형의 유용함은 둘 이상의 연관이 있는 이름을 상수로 선언함으로써  
프로그램의 가독성을 높이는데 있다.

# 질문 및 정리

