

The background of the slide is a composite image. The top left portion shows a globe of the Earth with a grid of latitude and longitude lines, set against a blue gradient. The bottom right portion shows a close-up of a hand using a calculator, with the keypad and display visible. A semi-transparent purple and orange horizontal bar spans across the middle of the image, containing the title text.

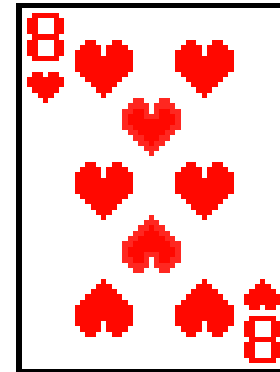
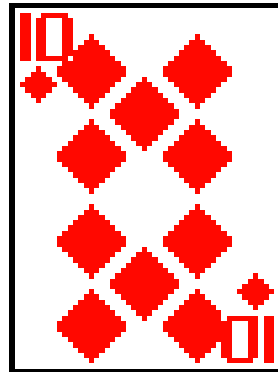
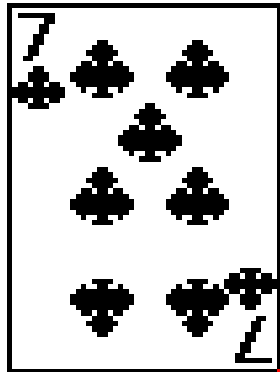
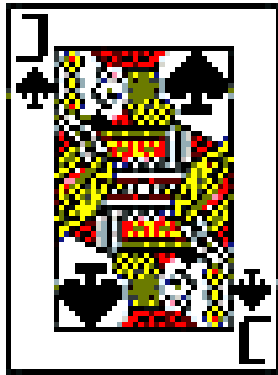
## 9장 구조체와 공용체

# 구조체와 공용체

- C 언어의 확장 방법
  - 매크로와 라이브러리
  - 사용자 정의 형 (배열, 구조체, 공용체)

# 구조체

- 서로 다른 형의 변수들을 하나로 묶어 주는 방법
- 예제 - 카드



# 구조체

- 예제 - 카드

- 각 카드는 고유의 무늬와 숫자를 가짐  
→ 구조체를 사용하여 표현하면 효율적
- 카드를 위한 구조체 선언

```
struct card {  
    int    pips;  
    char   suit;  
};
```

# 구조체 선언

- 예제 - 카드

```
struct card {  
    int    pi ps;  
    char   sui t;  
};
```

- **struct** : 키워드

- **card** : 구조체 태그 이름

- **pi ps, sui t** : 구조체 멤버

\* 이것은 **struct card** 형의 정의이고, 변수 선언은 아님

# 구조체 변수 선언 방법 1

- **struct card** 형 변수 **c1, c2**

```
struct card {  
    int    pi ps;  
    char   sui t;  
};  
struct   card    c1, c2;
```

## 구조체 변수 선언 방법 2

- **struct card** 형 변수 **c1, c2**

```
struct card {  
    int    pi ps;  
    char  sui t;  
} c1, c2;
```

# 구조체 변수 선언 방법 3

- **struct card** 형 변수 **c1, c2**

```
struct card {  
    int    pi ps;  
    char   sui t;  
};  
typedef   struct card   card;  
card     c1, c2;
```



- **struct card** 형 변수 **c1, c2**

```
typedef struct {  
    int    pi ps;  
    char   sui t;  
} card;  
card  c1, c2;
```

# 구조체 변수 선언 방법 5

- **struct** 형 변수 선언

```
struct {  
    int    pi ps;  
    char   sui t;  
} c1, c2;
```

- 구조체 태그 이름이 없음에 주의
- **C1**, **c2**와 같은 형의 변수는 다시는 선언할 수 없음

# 구조체 변수 선언 방법

```
struct {  
    int    pi ps;  
    char   sui t;  
} c1, c2;
```

```
struct {  
    int    pi ps;  
    char   sui t;  
} c3, c4;
```

\* c1, c2는 c3, c4와는 다른 형임

# 구조체 멤버

- 한 구조체내에서 멤버 이름은 유일해야 하나, 서로 다른 구조체에서는 같은 멤버 이름을 사용할 수 있음

```
struct fruit {  
    char    *name;  
    int     calories;  
};  
struct vegetable {  
    char    *name;  
    int     calories;  
};  
struct fruit    a;  
struct vegetable b;
```

# 멤버 접근 연산자 .

- 일반 구조체 멤버 접근 연산자

```
c1. pi ps = 3;
```

```
c1. sui t = 's' ;
```

```
c2. pi ps = c1. pi ps;
```

```
c2. sui t = c1. sui t;
```

# 멤버 접근 연산자 , 예제

```
#define CLASS_SIZE 100
struct student {
    char *last_name;
    int student_id;
    char grade;
};
struct student class[CLASS_SIZE];
int fail(struct student class[]) {
    int i, cnt = 0;
    for (i = 0; i < CLASS_SIZE; ++i)
        cnt += class[i].grade == 'F';
    return cnt;
}
```

# 멤버 접근 연산자 ->

- 포인터를 통한 구조체 멤버 접근 연산자

`pointer_to_structure -> member_name`

- 다른 방법

`(*pointer_to_structure).member_name`

# 멤버 접근 연산자 -> 예제

```
struct complex {  
    double re;          /* real part */  
    double im;         /* imag part */  
};  
  
typedef struct complex complex;  
  
void add(complex *a, complex *b, complex *c) {  
    a -> re = b -> re + c -> re;  
    a -> im = b -> im + c -> im;  
}
```



# 멤버 접근 연산자

선언문과 배정문

```
struct student  tmp, *p = &tmp;
tmp.grade = 'A';
tmp.last_name = "Casanova";
tmp.student_id = 910017
```

수식	동등한 수식	개념적 값
<code>tmp.grade</code>	<code>p -&gt; grade</code>	<b>A</b>
<code>tmp.last_name</code>	<code>p -&gt; last_name</code>	<b>Casanova</b>
<code>(*p).student_id</code>	<code>p -&gt; student_id</code>	<b>910017</b>
<code>*p -&gt; last_name + 1</code>	<code>*(p -&gt; last_name) + 1</code>	<b>D</b>
<code>*(p -&gt; last_name + 2)</code>	<code>(p -&gt; last_name)[2]</code>	<b>s</b>

# 연산자의 우선 순위와 결합 법칙

연산자	결합 법칙
() [] . -> ++ (후위) -- (후위)	좌에서 우로
++ (전위) -- (전위) ! ~ sizeof (형) + (단항) - (단항) & (주소) * (역참조)	우에서 좌로
* / %	좌에서 우로
+ -	좌에서 우로
<< >>	좌에서 우로
< <= > >=	좌에서 우로
== !=	좌에서 우로
&	좌에서 우로
^	좌에서 우로
	좌에서 우로
& &	좌에서 우로
	좌에서 우로
?:	우에서 좌로
= += -= *= /= %= <<= >>= &= ^=  =	우에서 좌로
, (콤마 연산자)	좌에서 우로

# 함수에서 구조체

- 구조체는 함수의 인자로서 함수에 전달될 수 있고, 함수로부터 리턴될 수도 있음
- 함수의 인자로서 구조체가 전달될 때 구조체는 값으로 전달됨
- 구조체가 많은 멤버를 가지거나, 큰 배열을 멤버로 가질 경우, 함수의 인자로 구조체를 전달하는 것은 상대적으로 비효율적임
- 따라서 대부분의 응용 프로그램에서는 함수의 인자로 구조체의 주소를 사용함

# 함수에서 구조체 예제

```
typedef struct {  
    char        name[25];  
    int         employee_id;  
    struct dept department;  
    struct home_address *a_ptr;  
    double      salary;  
    .....  
} employee_data;
```

\* department 멤버는 그 자체가 구조체이고, 컴파일러는 각 멤버의 크기를 미리 알아야 하므로 `struct dept`에 대한 선언이 먼저 와야 함

# 함수에서 구조체 예제

- 함수 선언 방법 1

```
employee_data update(employee_data e) {  
    .....  
    printf("Input the department number: ");  
    scanf("%d", &n);  
    e.department.dept_no = n;  
    .....  
    return e;  
}
```

\* 함수 호출

```
employee_data e;  
e = update(e);
```

# 함수에서 구조체 예제

- 함수 선언 방법 2

```
Void update(employee_data *p) {  
    .....  
    printf("Input the department number: ");  
    scanf("%d", &n);  
    p->department.dept_no = n;  
    .....  
}
```

\* 함수 호출

```
employee_data e;  
update(&e);
```

# 구조체의 초기화

```
card      c = {13, 'h'};          /* the king of hearts */
complex  a[3][3] = {
    {{1.0, -0.1}, {2.0, 0.2}, {3.0, 0.3}},
    {{4.0, -0.4}, {5.0, 0.5}, {6.0, 0.6}},
};      /* a[2][] is assigned zeroes */
struct fruit  frt = {"plum", 150};
struct home_address {
    char      *street;
    char      *city_and_state;
    long      zip_code;
} address = {"87 West Street", "Aspen, Colorado", 80526};
struct home_address  previous_address = {0};
```

# 공용체

- union
- 공용체는 구조체와 비슷한 구문 형식을 가지지만 각 멤버들은 같은 기억장소를 공유함
- 공용체형은 메모리의 같은 위치에 저장될 여러 값의 집합을 정의
- 저장된 값을 올바르게 해석하는 것은 프로그래머의 책임



# 공용체 선언

- 예제

```
union int_or_float {  
    int    i;  
    float  f;  
};
```

- union : 키워드

- int\_or\_float : 공용체 태그 이름

- i, f : 공용체 멤버

- \* int\_or\_float 형 변수는  $\text{MAX}(\text{sizeof}(\text{int}), \text{sizeof}(\text{float}))$  만큼의 메모리 할당될 것임

# 공용체 변수 선언

- 공용체 변수 선언

```
union int_or_float    a, b, c;
```

- 이 선언으로 식별자 a, b, c에 대한 기억장소가 할당

- 컴파일러는 공용체의 멤버 중에서 가장 큰 기억장소를 요구하는 멤버의 요구만큼 기억장소를 할당
- 공용체의 멤버 접근 방법은 구조체의 멤버 접근 방법과 동일

# 공용체 예제

```
int main(void) {
    union int_or_float    n;
    n.i = 4444;
    printf("i: %10d      f: %16.10e\n", n.i, n.f);
    n.f = 4444.0;
    printf("i: %10d      f: %16.10e\n", n.i, n.f);
    return 0;
}
```

- 출력 결과 예:

```
    i:          4444      f: 0.6227370375e-41
    i: 1166729216      f: 4.4440000000e+03
```

# 비트 필드

- 구조체나 공용체에서 `int` 형이나 `unsigned` 형의 멤버에 비트 수(**폭**)를 지정하는 것
- 폭은 콜론 다음에 음수가 아닌 **정수적형 상수 수식**으로 지정되고, 최대 값은 기계 워드의 비트 수와 같음
- 일반적으로 비트 필드는 구조체의 연속적인 멤버로 선언되며, 컴파일러는 이 멤버들을 최소의 기계 워드로 패킹함
- `unsigned` 비트 필드에는 음수가 아닌 정수만이 저장되고, `int` 비트 필드는 시스템에 따라 다름 - 보통 `unsigned` 비트 필드를 사용

# 비트 필드 예제

- 카드는 4개의 무늬와 각 무늬당 13개의 숫자로 이루어짐
- 카드를 비트 필드로 표현하면 메모리를 절약할 수 있음

```
struct pcard {  
    unsigned    pi ps : 4;  
    unsigned    sui t : 2;  
};
```

```
struct pcard    c;
```

\* c는 6 비트에 저장됨

# 비트 필드 예제

- 대부분의 컴퓨터에서는 비트 필드가 워드 경계에 걸치지 않도록 할당됨
- 예제

```
struct abc {  
    int    a : 1, b : 16, c : 16;  
} x;
```

- \* 이 경우 x는 두 워드에 다음과 같이 저장됨
  - 첫 번째 워드 : 비트 필드 a와 b 저장
  - 두 번째 워드 : c 저장

# 비트 필드 제약사항

- 비트 필드의 배열은 허용되지 않음
- 비트 필드에 주소 연산자 & 를 적용할 수 없음
  - 즉, 멤버 접근 연산자 -> 를 사용할 수는 있어도,  
포인터가 직접 비트 필드를 포인팅할 수는 없음

# 비트 필드

- 패딩과 정렬을 위해 이름 없는 비트 필드나 폭이 0인 비트 필드를 사용할 수 있음

```
struct small_integers {
    unsigned    i1 : 7, i2 : 7, i3 : 7,
                : 11, /* align to next word */
                i4 : 7, i5 : 7, i6 : 7;
}

struct abc {
    unsigned    a : 1, : 0, b : 1, : 0, c : 1;
};
```



# 비트와 바이트의 접근

- 비트와 바이트를 접근하기 위해 다음과 같이 선언할 수 있음

```
typedef struct {
    unsigned    b0 : 8, b1 : 8, b2 : 8, b3 : 8;
} word_bytes;

typedef struct {
    unsigned
        b0 : 1, b1 : 1, b2 : 1, b3 : 1, b4 : 1, b5 : 1, b6 : 1,
        b7 : 1, b8 : 1, b9 : 1, b10 : 1, b11 : 1, b12 : 1, b13 : 1,
        b14 : 1, b15 : 1, b16 : 1, b17 : 1, b18 : 1, b19 : 1, b20 : 1,
        b21 : 1, b22 : 1, b23 : 1, b24 : 1, b25 : 1, b26 : 1, b27 : 1,
        b28 : 1, b29 : 1, b30 : 1, b31 : 1;
} word_bits;

typedef union {
    int            i;
    word_bits      bit;
    word_bytes     byte;
} word;
```

# 비트와 바이트의 접근

```
int main(void)
{
    word    w = {0};
    void    bit_print(int);
    w.bit.b8 = 1;
    w.byte.b0 = 'a';
    printf("w.i = %d\n", w.i);
    bit_print(w.i);
    return 0;
}
```

**w.i = 353**

**00000000 00000000 00000001 01100001**

**w.i = 1635778560**

**01100001 10000000 00000000 00000000**

# 접시 보관 방법



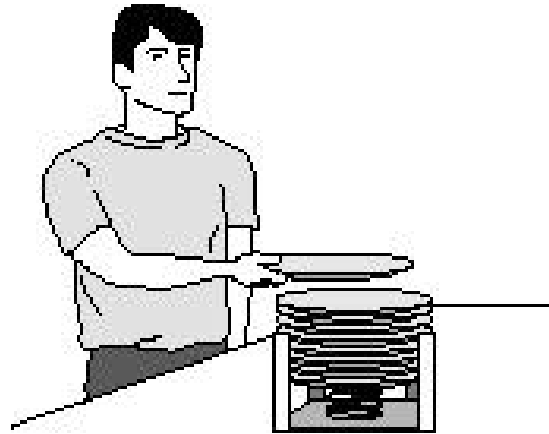
- 자료구조 (접시 보관장, 접시 개수대, 찬장)
- 연산 (접시 넣기, 접시 꺼내기)

# ADT

- ADT (Abstract Data Type: 추상 자료형)
  - 연산자를 포함한 자료구조
  - 구현에 대해서는 고려하지 않음

# 스택

- 자료 저장을 위한 자료구조로 자료의 삽입과 삭제가 톱이라는 스택의 한쪽 끝에서 일어남
- 삭제는 후입선출 방법으로 일어남



- 스택 연산 : push, pop, top, empty, full, reset
- ADT 스택

# ADT 스택 구현

- 스택 구현 방법
  - 배열
  - 선형 연결 리스트

# ADT 스택 구현 예제

```
#define MAX_LEN 1000
#define EMPTY -1
#define FULL (MAX_LEN - 1)

typedef enum boolean {false, true} boolean;
typedef struct stack {
    char s[MAX_LEN];
    int top;
} stack;
```

# ADT 스택 구현 예제

```
void reset(stack *stk) {  
    stk -> top = EMPTY;  
}
```

```
void push(char c, stack *stk) {  
    stk -> top++;  
    stk -> s[stk -> top] = c;  
}
```

```
char pop(stack *stk) {  
    return (stk -> s[stk -> top--]);  
}
```



# ADT 스택 구현 예제

```
char top(const stack *stk) {  
    return (stk -> s[stk -> top]);  
}
```

```
boolean empty(const stack *stk) {  
    return ((boolean) (stk -> top == EMPTY));  
}
```

```
boolean full(const stack *stk) {  
    return ((boolean) (stk -> top == FULL));  
}
```