Structures and Unions

Jin-Soo Kim
(*jinsoo.kim@snu.ac.kr*)

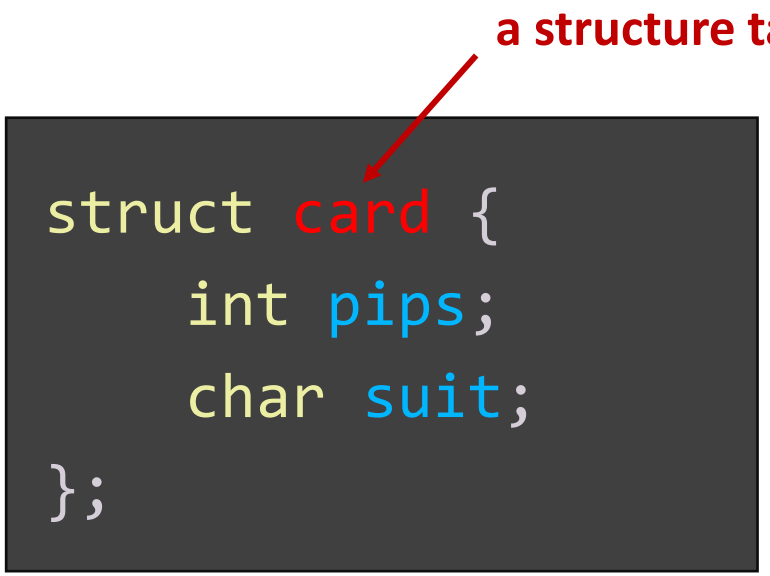
Systems Software &
Architecture Lab.

Seoul National University

Spring 2023

Structures

- Array
  - A derived type used to represent homogeneous data
- Structure
  - Provides a means to aggregate variables of different types



- **This declaration creates the derived data type `struct card`**
- **A user-defined type**
- **Just a template, no storage allocated**

Declaring Structures (1)

```
struct card {
    int pips;
    char suit;
};
struct card c1, c2;
```

```
struct card {
    int pips;
    char suit;
} c1, c2;
```

```
struct card {
    int pips;
    char suit;
};
typedef struct card card;
card c1, c2;
```

```
struct card {
    int pips;
    char suit;
} deck[52];
```

The identifier `deck` is declared to be an array of `struct card`

```
typedef struct {
    float re;
    float im;
} complex;
complex a, b, c[100];
```

When using `typedef` to name a structure type, the tag name may be unimportant

Declaring Structures (2)

- If a tag name is not supplied, then the structure type cannot be used in later declarations

```
struct {
    int day, month, year;
    char day_name[4];
    char month_name[4];
} yesterday, today, tomorrow;
```

vs.

```
struct date {
    int day, month, year;
    char day_name[4];
    char month_name[4];
};
struct date yesterday, today, tomorrow;
```

Member Access Operators

- Member access operator `.`
  - ***structure_variable*.*member_name***
  - `c1.pips = 3;`
  - `c1.suit = 's';`
- Member access operator `->`
  - Access the structure members via a pointer:
  - ***pointer_to_structure*->*member_name***
  - `struct card *c = &c1;`
  - `c->pips = 3;` ⇔ `(*c).pips = 3;` ≠ `*c.pips = 3;`
  - Note: the operators "`.`" and `->` have the highest precedence (from left to right)
- Structure assignment: `c1 = c2`

Operator Precedence & Associativity

Operator	Associativity
`()` `[]` `.` `->` `++` (*postfix*) `--` (*postfix*)	Left to right
`+` (*unary*) `-` (*unary*) `++` (*prefix*) `--` (*prefix*) `!` `&` (*address*) `*` (*dereference*) `~`	Right to left
`*` `/` `%`	Left to right
`+` `-`	Left to right
`<<` `>>`	Left to right
`<` `<=` `>` `>=`	Left to right
`==` `!=`	Left to right
`&`	Left to right
`^`	Left to right
`|`	Left to right
`&&`	Left to right
`||`	Left to right
`?:`	Right to left
`=` `+=` `-=` `*=` `/=` `%=` `<<=` `>>=` `&=` `^=` `|=`	Right to left
`,` (*comma operator*)	Left to right

Structure Members

- Within a given structure, member names must be unique

- Members in different structures can have the same name

```
struct fruit {
    char *name;
    int calories;
};

struct vegetable {
    char *name;
    int calories;
};

struct fruit a;
struct vegetable b;

a.calories = 100;
b.calories = 120;
```

Example 1

class_info.h

```
#define CLASS_SIZE      100
struct student {
    char *last_name;
    int  student_id;
    char grade;
};
```

grade.c

```
#include "class_info.h"
int main(void) {
    struct student tmp, class[CLASS_SIZE];

    tmp.grade = 'A';
    tmp.last_name = "Hong";
    tmp.student_id = 123456;

    ...
}
```

Example 1 (cont'd)

grade_fail.c

```c
#include "class_info.h"

/* Count the failing grades */
int fail(struct student class[])        /* int fail(struct student *class) */
{
    int i, cnt = 0;

    for (i = 0; i < CLASS_SIZE; i++)
        cnt += class[i].grade == 'F';   /* cnt += (class[i].grade == 'F'); */

    return cnt;
}
```

Example 2

complex.h

```c
struct complex {
    double re;
    double im;
};
typedef struct complex complex;
```

add.c

```c
#include "complex.h"

void add(complex *a, complex *b, complex *c)
{
    a->re = b->re + c->re;
    a->im = b->im + c->im;
}
```

Structure Initialization

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

card c = {13, 'h'};
```

```c
typedef struct {
    float re;
    float im;
} complex;

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 */
```

Accessing Members of a Structure

Declarations and initializations

```
struct student {
    char *last_name;
    int student_id;
    char grade;
};
struct student tmp = {"Hong", 123456, 'A' };
struct student *p = &tmp;
```

Expression	Equivalent expression	Action
`tmp.grade`	`p->grade`	`'A'`
`tmp.last_name`	`p->last_name;`	`"Hong"`
`(*p).student_id`	`tmp.student_id`	`123456`
`*p->last_name - 1`	`(*(p->last_name)) - 1`	`'G'`
`*(p->last_name + 2)`	`(p->last_name)[2]`	`'n'`

Using Structures with Functions (1)

- When a structure is passed as an argument to a function, it is passed by value
  - A local copy is made for use in the body of the function
  - If a structure member is an array, the array gets copied as well
  - Relatively inefficient!!

```
struct dept {
    char dept_name[25];
    int  dept_no;
};

typedef struct {
    char name[25];
    int  employee_id;
    struct dept department;
    double salary;
} employee;
```

Using Structures with Functions (2)

```c
employee update(employee r)
{
    printf("Input dept. number: ");
    scanf("%d", &n);
    r.department.dept_no = n;

    return r;
}

int main(void) {
    employee e;

    ...

    e = update(e);
}
```

```c
void update(employee *p)
{
    printf("Input dept. number: ");
    scanf("%d", &n);
    p->department.dept_no = n;

    return r;
}

int main(void) {
    employee e;

    ...

    update(&e);
}
```

Unions (1)

- A union defines a set of alternative values that may be stored in a shared portion of memory
  - A derived type, following the same syntax as the structures
  - Union members share storage
  - The compiler allocates a piece of storage that can accommodate the largest of members

```c
#include <stdio.h>

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

```c
int main(void)
{
    union intfloat a;

    a.f = 1.0;
    printf("a.i = %#x\n", a.i);
}
```

Unions (2)

- Bit fields
  - An int or unsigned member of a structure or a union can be declared to consist of a specified number of bits, i.e., a bit field member
  - Width (# of bits) is specified by a nonnegative constant integer following a colon :

```
typedef struct {
    unsigned b0:8;
    unsigned b1:8;
    unsigned b2:8;
    unsigned 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;
```

Unions (3)

```c
typedef union {
    int         i;
    word_bits   bit;
    word_bytes  byte;
    char        ch[4];
} word;

int main(void) {
    word w = {0};

    w.bit.b0 = 1;
    w.byte.b3 = 0xff;
    printf("%#x\n", w.i);

    w.i = 0x12345678;
    printf("%x %x %x %x\n", w.byte.b0, w.byte.b1, w.byte.b2, w.byte.b3);
    printf("%x %x %x %x\n", w.ch[0], w.ch[1], w.ch[2], w.ch[3]);
}
```

Playing Poker (1)

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <assert.h>

#define NPIPS       13      /* number of pips */
#define NCARDS      52      /* total number of cards */
#define NDEALS      3000    /* number of deals */
#define NPLAYERS    6       /* number of players */
#define NHANDS      5       /* each player got 5 cards */

typedef enum {clubs, diamonds, hearts, spades, END} cdhs;

typedef struct {
    int pips;
    cdhs suit;
} card;
```

Playing Poker (2)

```
void init_deck(card *deck)
{
    int  pips;
    cdhs suit;
    int n = 0;

    for (pips = 1; pips <= NPIPS; pips++)
        for (suit = clubs; suit < END; suit++)
        {
            deck[n].pips = pips;
            deck[n++].suit = suit;
        }
}
```

Playing Poker (3)

```
void swap(card *p, card *q)
{
    card tmp;

    tmp = *p;
    *p = *q;
    *q = tmp;
}

void shuffle(card *deck)
{
    int i, j;
    for (i = 0; i < NCARDS; i++)
    {
        j = rand() % NCARDS;
        swap(&deck[i], &deck[j]);
    }
}
```

Playing Poker (4)

```
void deal_the_cards(card *deck, card hand[][NHANDS])
{
    int card_cnt = 0, i, j;

    for (j = 0; j < NHANDS; j++)
        for (i = 0; i < NPLAYERS; i++)
            hand[i][j] = deck[card_cnt++];
}

int is_flush(card hand[])
{
    int i;

    for (i = 1; i < NHANDS; i++)
        if (hand[0].suit != hand[i].suit)
            return 0;
    return 1;
}
```

Playing Poker (5)

```
void play_poker(card *deck) {
    int i, j;
    int flush_cnt = 0, hand_cnt = 0;
    card hand[NPLAYERS][NHANDS];     /* each player dealt 5 cards */

    srand(time(NULL));               /* seed random-number generator */
    for (i = 0; i < NDEALS; i++) {
        shuffle(deck);
        deal_the_cards(deck, hand);
        for (j = 0; j < NPLAYERS; j++) {
            hand_cnt++;
            if (is_flush(hand[j]))
                flush_cnt++;
        }
    }
    printf("Flush probability: %d / %d = %f \n",
           flush_cnt, hand_cnt, (double) flush_cnt / hand_cnt);
}
```

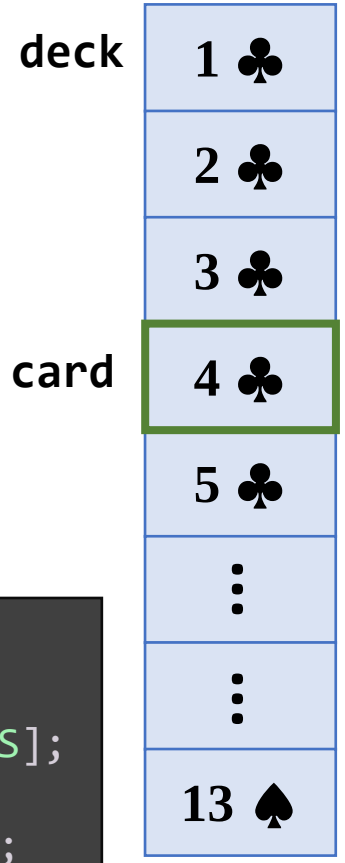
Playing Poker (6)

```c
int main(void)
{
    card deck[NCARDS];

    init_deck(deck);
    play_poker(deck);
}
```

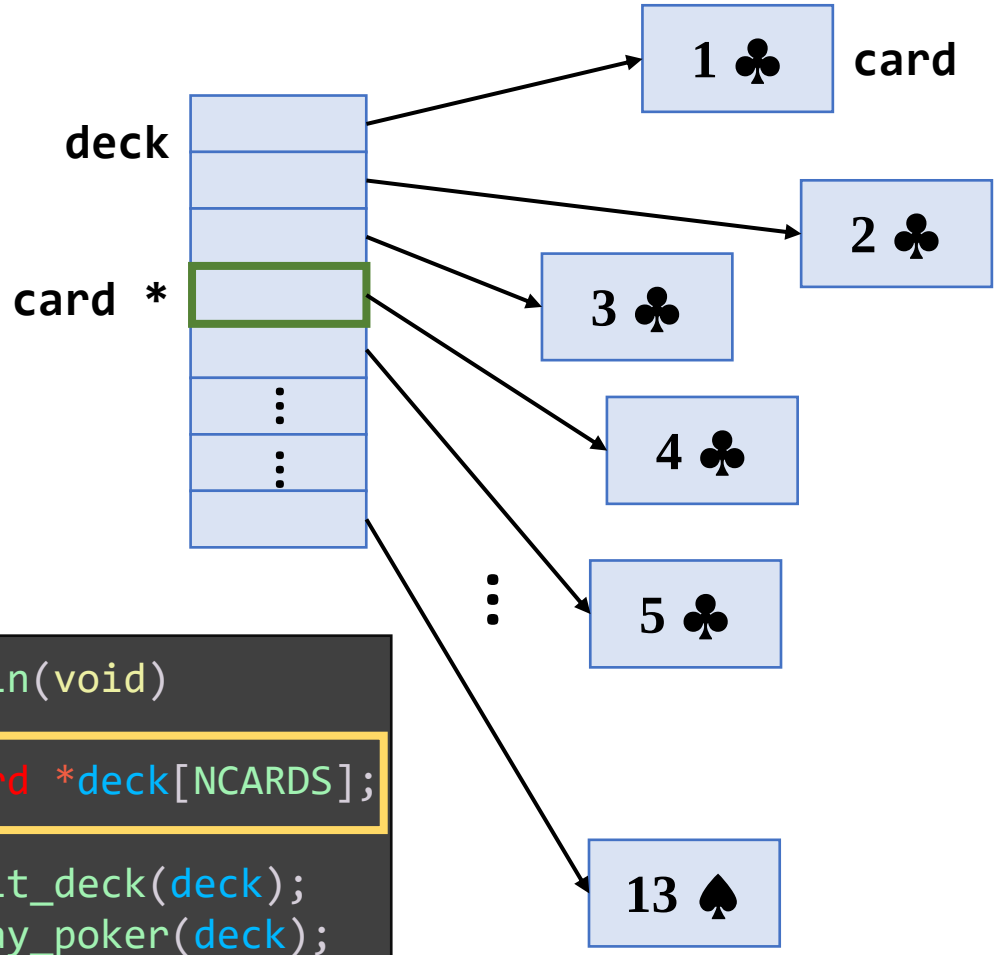
```
$ gcc poker.c
$ ./a.out
Flush probability: 35 / 18000 = 0.001944
$ ./a.out
Flush probability: 32 / 18000 = 0.001778
$ ./a.out
Flush probability: 33 / 18000 = 0.001833
$ ./a.out
Flush probability: 31 / 18000 = 0.001722
```

Representing a Card Deck




```
int main(void)
{
    card deck[NCARDS];

    init_deck(deck);
    play_poker(deck);
}
```




```
int main(void)
{
    card *deck[NCARDS];

    init_deck(deck);
    play_poker(deck);
}
```

Playing Poker 2 (1)

```
void init_deck(card *deck[])
{
    int  pips;
    cdhs suit;
    int n = 0;

    for (pips = 1; pips <= NPIPS; pips++)
        for (suit = clubs; suit < END; suit++)
        {
            deck[n] = (card *) malloc(sizeof(card));
            assert(deck[n] != NULL);
            deck[n]->pips = pips;
            deck[n++]->suit = suit;
        }
}
```

Playing Poker 2 (2)

```
void swap(card **p, card **q)
{
    card *tmp;

    tmp = *p;
    *p = *q;
    *q = tmp;
}

void shuffle(card *deck[])
{
    int i, j;
    for (i = 0; i < NCARDS; i++)
    {
        j = rand() % NCARDS;
        swap(&deck[i], &deck[j]);
    }
}
```

Playing Poker 2 (3)

```
void deal_the_cards(card *deck[], card hand[][NHANDS])
{
    int card_cnt = 0, i, j;

    for (j = 0; j < NHANDS; j++)
        for (i = 0; i < NPLAYERS; i++)
            hand[i][j] = *(deck[card_cnt++]);
}

int is_flush(card hand[])
{
    int i;

    for (i = 1; i < NHANDS; i++)
        if (hand[0].suit != hand[i].suit)
            return 0;
    return 1;
}
```

Playing Poker 2 (4)

```
void play_poker(card *deck[]) {
    int i, j;
    int flush_cnt = 0, hand_cnt = 0;
    card hand[NPLAYERS][NHANDS];     /* each player dealt 5 cards */

    srand(time(NULL));           /* seed random-number generator */
    for (i = 0; i < NDEALS; i++) {
        shuffle(deck);
        deal_the_cards(deck, hand);
        for (j = 0; j < NPLAYERS; j++) {
            hand_cnt++;
            if (is_flush(hand[j]))
                flush_cnt++;
        }
    }
    printf("Flush probability: %d / %d = %f \n",
           flush_cnt, hand_cnt, (double) flush_cnt / hand_cnt);
}
```