
C Programming for Engineers

Structures, Unions



UNIVERSITY
AT ALBANY
State University of New York

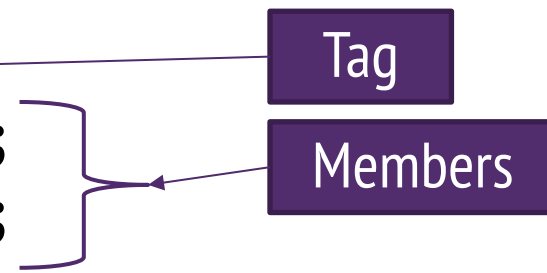
ICEN 360– Spring 2017

Prof. Dola Saha

Structure

- Collections of related variables under one name.
- Variables of may be of different data types.

➤ **struct** card {
 char *face;
 char *suit;
};



- Keyword **struct** introduces the structure definition.
- Members of the same structure type must have unique names, but two different structure types may contain members of the same name without conflict.

Structure Declaration

- `struct employee {
 char firstName[20];
 char lastName[20];
 unsigned int age;
 char gender;
 double hourlySalary;
};`
- `struct employee employee1, employee2;`
- `struct employee employees[100];`
- `struct employee {
 char firstName[20];
 char lastName[20];
 unsigned int age;
 char gender;
 double hourlySalary;
} employee1, employee2, *employeePtr;`

Structure Tag

- The structure tag name is optional.
- If a structure definition does not contain a structure tag name, variables of the structure type may be declared *only* in the structure definition – *not* in a separate declaration.

Self Reference

- *A structure cannot contain an instance of itself.*
- A variable of type `struct employee` cannot be declared in the definition for `struct employee`.
- A pointer to `struct employee`, may be included.
- For example,
 - ```
struct employee2 {
 char firstName[20];
 char lastName[20];
 unsigned int age;
 char gender;
 double hourlySalary;
 struct employee2 person; // ERROR
 struct employee2 *ePtr; // pointer
};
```
- `struct employee2` contains an instance of itself (`person`), which is an error.

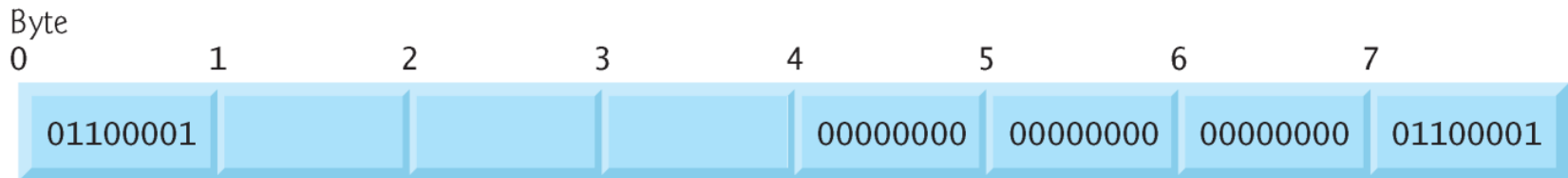
# Storage in Memory

---

- Structures *may not* be compared using operators == and !=, because
  - structure members are *not necessarily stored in consecutive bytes of memory*.
- Computers may store specific data types only on certain memory boundaries such as half-word, word or double-word boundaries.
- A word is a standard memory unit used to store data in a computer—usually 2 bytes or 4 bytes.

# Storage in Memory

```
➤ struct example {
 char c;
 int i;
} sample1, sample2;
```



Possible storage, but machine dependant

# Initialization

---

- **struct** card {  
    **char** \*face;  
    **char** \*suit;  
};
- **struct** card aCard = {"Three", "Hearts"};
- If there are fewer initializers in the list than members in the structure,
  - the remaining members are automatically initialized to 0
  - or NULL if the member is a pointer.
- Assignment Statement of same struct type
  - **struct** card aCard1 = aCard2;



# Accessing Structure Members

---

- the **structure member operator** (.)—also called the **dot operator**
  - `printf("%s", aCard.suit); // displays Hearts`
- the **structure pointer operator** (->)—also called the **arrow operator**.
  - `cardPtr = &aCard;`
  - `printf("%s", cardPtr->suit); // displays Hearts`
  - Following are equivalent
    - `cardPtr->suit`
    - `(*cardPtr).suit`

# Example

```
4 #include <stdio.h>
5
6 // card structure definition
7 struct card {
8 char *face; // define pointer face
9 char *suit; // define pointer suit
10 };
11
12 int main(void)
13 {
14 struct card aCard; // define one struct card variable
15
16 // place strings into aCard
17 aCard.face = "Ace";
18 aCard.suit = "Spades";
19
20 struct card *cardPtr = &aCard; // assign address of aCard to cardPtr
21
22 printf("%s%s\n%s%s\n%s%s\n", aCard.face, " of ", aCard.suit,
23 cardPtr->face, " of ", cardPtr->suit,
24 (*cardPtr).face, " of ", (*cardPtr).suit);
25 }
```

Ace of Spades  
Ace of Spades  
Ace of Spades

# Structure with Function

---

- Structures may be passed to functions by
  - passing individual structure members
  - by passing an entire structure
  - by passing a pointer to a structure.
- Functions can return
  - individual structure members
  - an entire structure
  - a pointer to a structure

# typedef

---

- The keyword **typedef** is a way to create synonyms (or aliases) for previously defined data types.
- Names for structure types are often defined with **typedef** to create shorter type names.
- Example:
  - **typedef struct** card Card;  
Card is a synonym for type **struct card**.
- Example:
  - **typedef struct** {  
    **char** \*face;  
    **char** \*suit;  
} Card;  
▪ Card myCard, \*myCardPtr, deck[52];

# Card Shuffling Example (1)

```
1 // Fig. 10.3: fig10_03.c
2 // Card shuffling and dealing program using structures
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <time.h>
6
7 #define CARDS 52
8 #define FACES 13
9
10 // card structure definition
11 struct card {
12 const char *face; // define pointer face
13 const char *suit; // define pointer suit
14 };
15
16 typedef struct card Card; // new type name for struct card
17
18 // prototypes
19 void fillDeck(Card * const wDeck, const char * wFace[],
20 const char * wSuit[]);
21 void shuffle(Card * const wDeck);
22 void deal(const Card * const wDeck);
23
```



# Card Shuffling Example (2)

---

```
24 int main(void)
25 {
26 Card deck[CARDS]; // define array of Cards
27
28 // initialize array of pointers
29 const char *face[] = { "Ace", "Deuce", "Three", "Four", "Five",
30 "Six", "Seven", "Eight", "Nine", "Ten",
31 "Jack", "Queen", "King"};
32
33 // initialize array of pointers
34 const char *suit[] = { "Hearts", "Diamonds", "Clubs", "Spades"};
35
36 srand(time(NULL)); // randomize
37
38 fillDeck(deck, face, suit); // load the deck with Cards
39 shuffle(deck); // put Cards in random order
40 deal(deck); // deal all 52 Cards
41 }
42
```



# Card Shuffling Example (3)

```
43 // place strings into Card structures
44 void fillDeck(Card * const wDeck, const char * wFace[],
45 const char * wSuit[])
46 {
47 // loop through wDeck
48 for (size_t i = 0; i < CARDS; ++i) {
49 wDeck[i].face = wFace[i % FACES];
50 wDeck[i].suit = wSuit[i / FACES];
51 }
52 }
53
54 // shuffle cards
55 void shuffle(Card * const wDeck)
56 {
57 // loop through wDeck randomly swapping Cards
58 for (size_t i = 0; i < CARDS; ++i) {
59 size_t j = rand() % CARDS;
60 Card temp = wDeck[i];
61 wDeck[i] = wDeck[j];
62 wDeck[j] = temp;
63 }
64 }
65
```



# Card Shuffling Example (4)

---

```
66 // deal cards
67 void deal(const Card * const wDeck)
68 {
69 // loop through wDeck
70 for (size_t i = 0; i < CARDS; ++i) {
71 printf("%5s of %-8s%s", wDeck[i].face, wDeck[i].suit,
72 (i + 1) % 4 ? " " : "\n");
73 }
74 }
```

---



# Card Shuffling Example (5)

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