Structures
Structures

- Structures
- Typedef
- Declarations
- Using Structures with Functions
10.1 Introduction

- Structures
  - A collection of related variables (aggregates) under one name.
    - Can contain variables of **different** data types.
  - Commonly used to define records to be stored in files.
  - When combined with pointers, structures can create linked lists, stacks, queues, and trees.
Example 1:
struct player
{
    char *name;
    int num;
    char *team;
    char *pos;
} ; /* Don't forget this semicolon! */
Example 1:
struct player
{
    char *name;
    int num;
    char *team;
    char *pos;
} player1, player2;
Example 2:

```c
struct card
{
    const char *face;
    const char *suit;
};
typedef struct card Card;
```

The new type `Card` is an alias for type `struct card`.

`struct` introduces the definition for structure `card`.
- `card` is the structure name and is used to declare variables of the structure type.
- `card` contains two members of type `char *`
  - These members are `face` and `suit`.
typedef

Another way::

typedef struct
{
    const char *face;
    const char *suit;
} Card;

...  
Card deck[52];
10.6 typedef

Example:

```
typedef struct Card *CardPtr;
```

or

```
Card *Cardptr;
```

- Defines a new type name `CardPtr` as an alias for type `struct Card *`.
- `typedef` does not create a new data type.
  - It only creates an alias.
- Capitalize the first letter of `typedef` names to emphasize that they are synonyms for other type names.
10.2 Structure Definitions

- **struct** information
  - A **struct** cannot contain an instance of itself.
  - It can contain a member that is a pointer to the same structure type (**a self-referential structure**).
  - A structure definition does not reserve space in memory. Rather a **struct** creates a new data type used to define structure variables.

- **Definitions**
  - Defined like other variables:
    ```c
card oneCard, deck[ 52 ], *cPtr;
```
  - Can use a comma separated list:
    ```c
    struct card {
        char *face;
        char *suit;
    } oneCard, deck[ 52 ], *cPtr;
    ```
10.2 Structure Definitions

- Valid Operations
  - Assigning a structure to a structure of the same type.
  - Taking the address (\&) of a structure
  - Accessing the members of a structure.
  - Using the `sizeof` operator to determine the size of a structure.
10.3 Initializing Structures

- **Initializer lists**
  - Example:
    ```
    struct card oneCard = { "Three", "Hearts" }; 
    ```

- **Assignment statements**
  - Example:
    ```
    struct card threeHearts = oneCard; 
    ```
  - Could also define and initialize `threeHearts` as follows:
    ```
    struct card threeHearts; 
    threeHearts.face = "Three"; 
    threeHearts.suit = "Hearts"; 
    ```
10.4 Accessing Members of Structures

- Accessing structure members
  - The dot operator (.) {the structure member operator} is used to access a structure member via the structure variable name.
    ```c
    card myCard;
    printf( "%s", myCard.suit );
    ```
  - The arrow operator (->) {the structure pointer operator} accesses a structure member via a pointer to the structure.
    ```c
    card *myCardPtr = &myCard;
    printf( "%s", myCardPtr->suit );
    ```
  - `myCardPtr->suit` is equivalent to `( *myCardPtr ).suit`
/ * Fig. 10.2: fig10_02.c
2 Using the structure member and
3 structure pointer operators */
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 }; /* end structure card */
11
12 int main( void )
13 {
14     struct card aCard; /* define one struct card variable */
15     struct card *cardPtr; /* define a pointer to a struct card */
16
17     /* place strings into aCard */
18     aCard.face = "Ace";
19     aCard.suit = "Spades";
cardPtr = &aCard; /* assign address of aCard to cardPtr */
printf("%s%s%s
%s%s%s
%s%s%s
", aCard.face, " of ", aCard.suit,
cardPtr->face, " of ", cardPtr->suit,
( *cardPtr ).face, " of ", ( *cardPtr ).suit );
return 0; /* indicates successful termination */
}

Arrow operator accesses members of a structure pointer
10.5 Using Structures with Functions

- Passing structures to functions
  - The entire structure can be passed.
  - Individual members of the structure can be passed.
  - For both cases, they are passed by value.

- To pass a structure by-reference
  - Pass the address of the structure variable.

- To pass arrays by-value
  - Create a structure with the array as a member and then pass the structure.
/* Fig. 10.3: fig10_03.c
The card shuffling and dealing program using structures */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/* card structure definition */
struct card {
    const char *face; /* define pointer face */
    const char *suit; /* define pointer suit */
}; /* end structure card */

typedef struct card Card; /* new type name for struct card */

/* prototypes */
void fillDeck( Card * const wDeck, const char * wFace[], 
    const char * wSuit[] );
void shuffle( Card * const wDeck );
void deal( const Card * const wDeck );

int main( void )
{
    Card deck[ 52 ]; /* 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"};
srand( time( NULL ) ); /* randomize */
fillDeck( deck, face, suit ); /* load the deck with Cards */
shuffle( deck ); /* put Cards in random order */
deal( deck ); /* deal all 52 Cards */
return 0; /* indicates successful termination */
} /* end main */

/* place strings into Card structures */
void fillDeck( Card * const wDeck, const char * wFace[],
const char * wSuit[] )
{
    int i; /* counter */
    /* loop through wDeck */
    for ( i = 0; i <= 51; i++ ) {
        wDeck[ i ].face = wFace[ i % 13 ];
        wDeck[ i ].suit = wSuit[ i / 13 ];
    } /* end for */
} /* end function fillDeck */
A Structure Example

/* shuffle cards */

void shuffle( Card * const wDeck )
{
    int i;  /* counter */
    int j;  /* variable to hold random value between 0 - 51 */
    Card temp;  /* define temporary structure for swapping Cards */

    /* loop through wDeck randomly swapping Cards */
    for ( i = 0; i <= 51; i++ ) {
        j = rand() % 52;
        temp = wDeck[ i ];
        wDeck[ i ] = wDeck[ j ];
        wDeck[ j ] = temp;
    } /* end for */

} /* end function shuffle */

/* deal cards */

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

    /* loop through wDeck */
    for ( i = 0; i <= 51; i++ ) {
        printf( "%s of %-8s%c", wDeck[ i ].face, wDeck[ i ].suit,
            ( i + 1 ) % 2 ? 't' : '\n' );
    } /* end for */

} /* end function deal */

Each card is swapped with another, random card, shuffling the deck.
A Structure Example

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

- Definition of structures in C
- Syntax details for declaring structs
- Initializing structs
- Typedef
- Structure member (.) and pointer -> operators
- Passing structures to functions
- A Structure Example