Structures

- Referred to as aggregates (related variables)
- Contain variables of many different types
- Commonly used to define records to be stored
- More complex data structures
- Self-referential `struct`
- Operations on `struct` instances
- Function pass by reference and pass by value
- `typedef`, `union`, `bitwise`, `bit fields`, `enum`

Define a Structure

- Structures are derived data types
- `struct` keyword
- `card` tag
- `face`, `card` members
- Semicolon must terminate a structure

```c
struct card {
  char *face;
  char *suit;
};
```

```
struct employee {
  char firstname[20];
  char lastname[20];
  unsigned int age;
  double hourlySalary;
};

struct employee2 {
  char firstname[20];
  char lastname[20];
  unsigned int age;
  double hourlySalary;
  struct employee2 person; // ERROR
  struct employee2 *emPtr; // Pointer
};
```

- Self-referential structure contains a member that’s a pointer to the same structure type. *(Linked Data)*

Structure Memory

- Structure definitions do not reserve any space in memory
- Creates a new data type used to define variables

```
struct card {
  char *face;
  char *suit;
};
struct card aCard, deck[52], *cardPtr;

// or
struct {
  char *face;
  char *suit;
} aCard, deck[52], *cardPtr;
```

Operations on Structures

- Assigning structure variables to structure variables
- Taking the address (&) of a structure variable
- Accessing the members of a structure variable
- Using the size of operator
- **Cannot** compare structures using == and !=
- Storage is not sequential or predictable dependent
- sizeof() returns total bytes occupied by the structure

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

Initializing Structures

- Missing information from initializing will be 0 or NULL
- Operators use to access members
  - Structure Member Operator .
  - Structure Pointer Operator ->

```c
struct card aCard = {"Three", "Hearts"};
// or
struct card aCard;
aCard.face = "Three";
aCard.suit = "Hearts";
printf("%s", aCard.suit);

struct card *cardPtr = &aCard;
printf("%s", aCard->suit);
printf("%s", (*cardPtr).suit);
```

```
include <stdio.h>

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

int main() {
   struct card aCard;
   struct card *cardPtr = &aCard;

   aCard.fact = "Ace";
   aCard.suit = "Spade";

   printf("%s of %s . Operator\n%s of %s -> Operator\n
   %s of %s Pointer", aCard.face, aCard.suit, aCard->face
   aCard->suit, (*cardPtr).face, (*cardPtr).suit);

   return 0;
}
```

Structures with Functions

- Passed by value when passing structure or individual members
    - Involves copying the entire contents of a structure
- Passed by reference requires the address of the structure
    - define pointers to structures to avoid copying overhead

Structure Tips and Errors

- Omit spaces around -> and . operators
- Spaces change the operators and cause a syntax error
- Refer to a structure member by name is a syntax error
- Modifying structure values in a function is a logic error
- Passing structures by reference is more efficient
- Pointers need () the structure pointer for dereference

Defining New Data Types with `typedef`

- Provides a mechanism for creating synonyms (or aliases)
- Can be used with structures
- Shortens the declaration of a structure
- Structure tag can be removed

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

Cards deck[52];
struct card deck2[52];
```

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

#define CARDS 52
#define FACES 13

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

void fillDeck(Card * const Deck,
    const char * Face[], const char * Suit[]);
void shuffle(Card * const Deck);
void deal(Card * const Deck);
void main() {
   Card deck[CARDS];
   const char *face[] = {"Ace", "Two",
    "Three", "Four", "Five", "Six",
     "Seven", "Eight", "Nine", "Ten",
     "Jack", "Queen", "King"};
   const char *suit[] = {"Hearts",
    "Diamonds", "Clubs", "Spades"};
   srand(time(NULL));

   fillDeck(deck, face, suit);
   shuffle(deck);
   deal(deck);
}
```

```c
void fillDeck(Card * const Deck, const char * Face[],
 const char * Suit[]) {
    for (size_t i = 0; i < CARDS; i++) {
        Deck[i].face = Face[i % FACES];
        Deck[i].suit = Suit[i / FACES];
    }
}

void shuffle(Card * const Deck) {
    for (size_t i = 0; i < CARDS; i++) {
        size_t j = rand() % CARDS;
        Card temp = Deck[i];
        Deck[i] = Deck[j];
        Deck[j] = temp;
    }
}
```

```c
void deal(Card * const Deck) {
    for (size_t i = 0; i < CARDS; i++) {
        printf("%5s of %-8s", Deck[i], Deck[i].suit);
        ((i+1)% 5) ? printf(" ") : printf("\n");
    }
    printf("\n");
}
```

Questions?