COP 3223: C Programming
Spring 2009

Structures In C – Part 1

Instructor : Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790
http://www.cs.ucf.edu/courses/cop3223/spr2009/section1

School of Electrical Engineering and Computer Science
University of Central Florida

Structures In C

- Structures – sometimes referred to as aggregates – are collections of related variables under one name.
- So far, we've only looked at one structure in C, the array. Arrays have two important properties that distinguish them from most structures. First, all array elements are of the same type. Second, the elements of the array are stored in contiguous locations in memory which allows us to specify a position in the structure using an index value (recall pointer arithmetic).
- The properties of a structure are quite different from that of an array. The elements (called members in C) are not required to have the same type, and the members of a structure each have a name, so to select a member of a structure its name is used not its position.

Structures In C

- Many programming languages have facilities for user defined structures. It is common in many languages, other than C, to refer to these structures as records, and the members of the records are called fields or attributes.
- It is common to used structures to define records that are stored in files.
- Pointers and structures are used to facilitate the formation of more complex data structures such as linked lists, queues, stacks, and trees. All of which are extensively used data structures in many computer science applications.
- Structures are considered to be derived data types, meaning that they are constructed using objects of other types.

Structures In C

- In C you declare a `struct` (essentially a type) and then you can create variables of the `struct` type.
- The general syntax of a `struct` declaration is:

```
struct <struct_name> {
        <type1> <variable1>;
        <type2> <variable2>;
        . . .
        <typeN> <variableN>;
};
```

Note: C convention is to place structure definitions at the top of your source file right after any #define directives.

Variables of the structure type can be declared by placing a comma-separated list between the closing brace of the structure definition and the semicolon that ends the structure definition.

Structures In C

- As an example structure declaration, let's create a structure that would contain information about students at UCF. We want to include the student's name, their GPA, and the number of credit hours they have completed. We might declare the structure as follows:

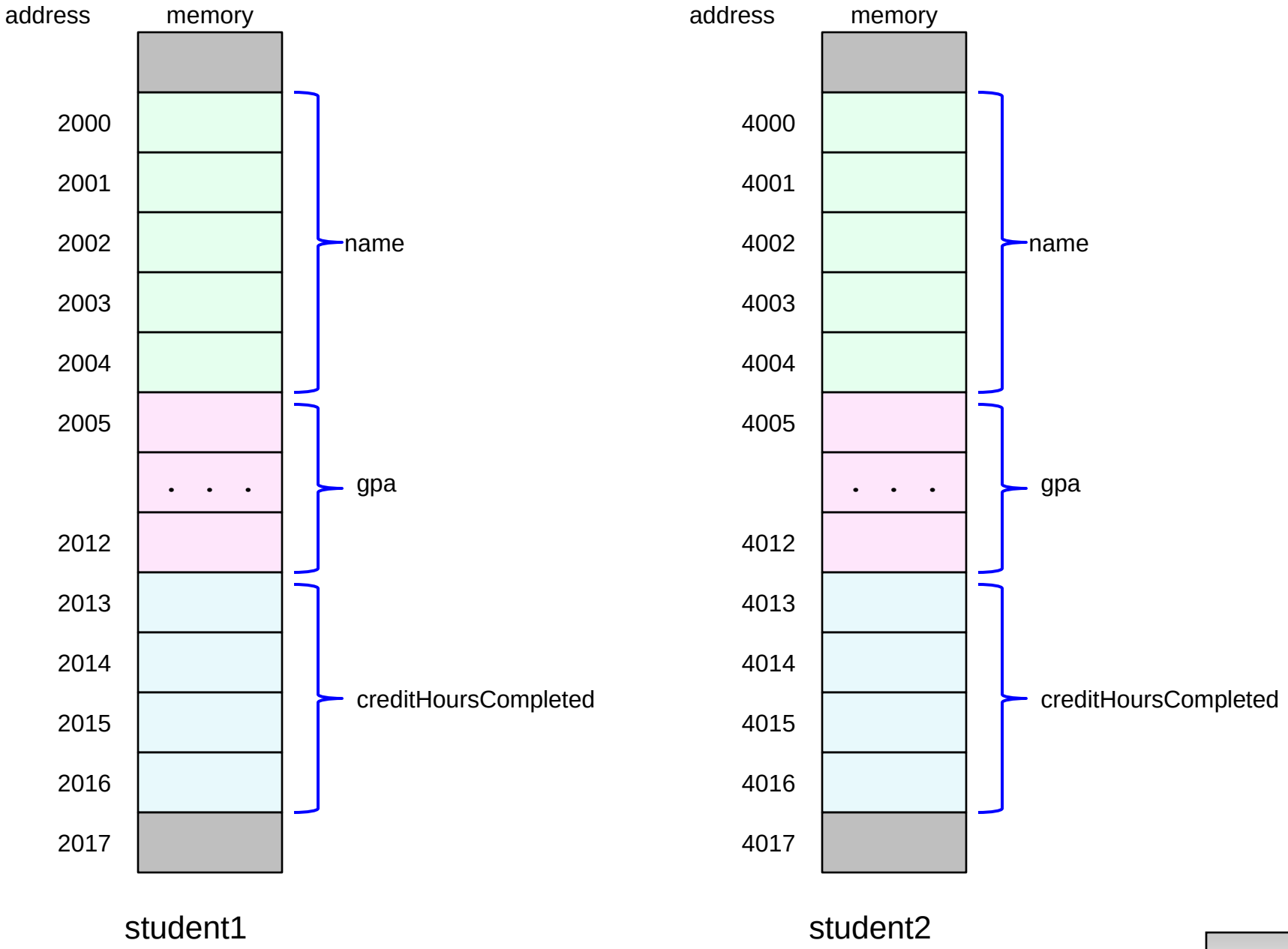
```
struct ucfStudent {
        char name[MAXLENGTH];
        double gpa;
        int creditHoursCompleted;
}student1, student2;
```

- This structure definition creates two variables named `student1` and `student2` with the structure as shown.

Structures In C

- The two variables `student1` and `student2` each have three members (fields) named `name`, `gpa`, and `creditHoursCompleted`.
- The members of a structure are stored in memory in the order in which they are declared. Assuming that `student1` is located at address 2000 in the memory and `student2` is located at address 4000 in memory, these structures in memory would be represented as shown in the diagram on the following page:

Let's assume that MAXLENGTH is 5, doubles require 8 bytes of memory and int requires 4 bytes of memory


address
memory
2000
2001
2002
2003
2004
2005
2012
2013
2014
2015
2016
2017
name
gpa
creditHoursCompleted
student1
address
memory
4000
4001
4002
4003
4004
4005
4012
4013
4014
4015
4016
4017
name
gpa
creditHoursCompleted
student2

Structures In C

- As with arrays, a structure variable can be initialized at the time it is declared. The initializer values must appear in the same order in which their corresponding members were declared in the structure and are enclosed in braces as they were with array initializers.

```
struct ucfStudent {
  char name[MAXLENGTH];
  double gpa;
  int creditHoursCompleted;
}student1 = {"Debi", 3.99, 110},
 student2 = {"Suzie", 3.25, 58};
```


Structures In C

- Structures may not be compared using the == or != operators, because structure members may not necessarily be stored in contiguous bytes of memory. Sometimes there are "holes" in a structure, because computers may store specific data types only on certain memory boundaries such as half word, word, or double word boundaries.
- For example, consider a computer with a 4-byte word and the structure definition:

```
struct example {
   char c;
   int n;
} sample1;
```

If sample1 were stored beginning at address 1000 (a word boundary) and its first member requires only 1 byte, the next word boundary would be at address 1004, leaving a hole of 3 bytes.

Accessing Members Of Structures

- The dot operator has the same precedence as the postfix ++ and – operators, which means it takes precedence over nearly all C operators (only () and [] are higher). C provides two operators for accessing the members of a structure.
- The structure member operator (`.`), more commonly called the dot operator. And the structure pointer operator (`->`). The structure pointer operator is used to access a structure member via a pointer to the structure. We'll use this operator later and for now focus only on the dot operator.
- To access a member of a structure use the following syntax:

```
nameOfTheStructure.nameOfTheMember
```

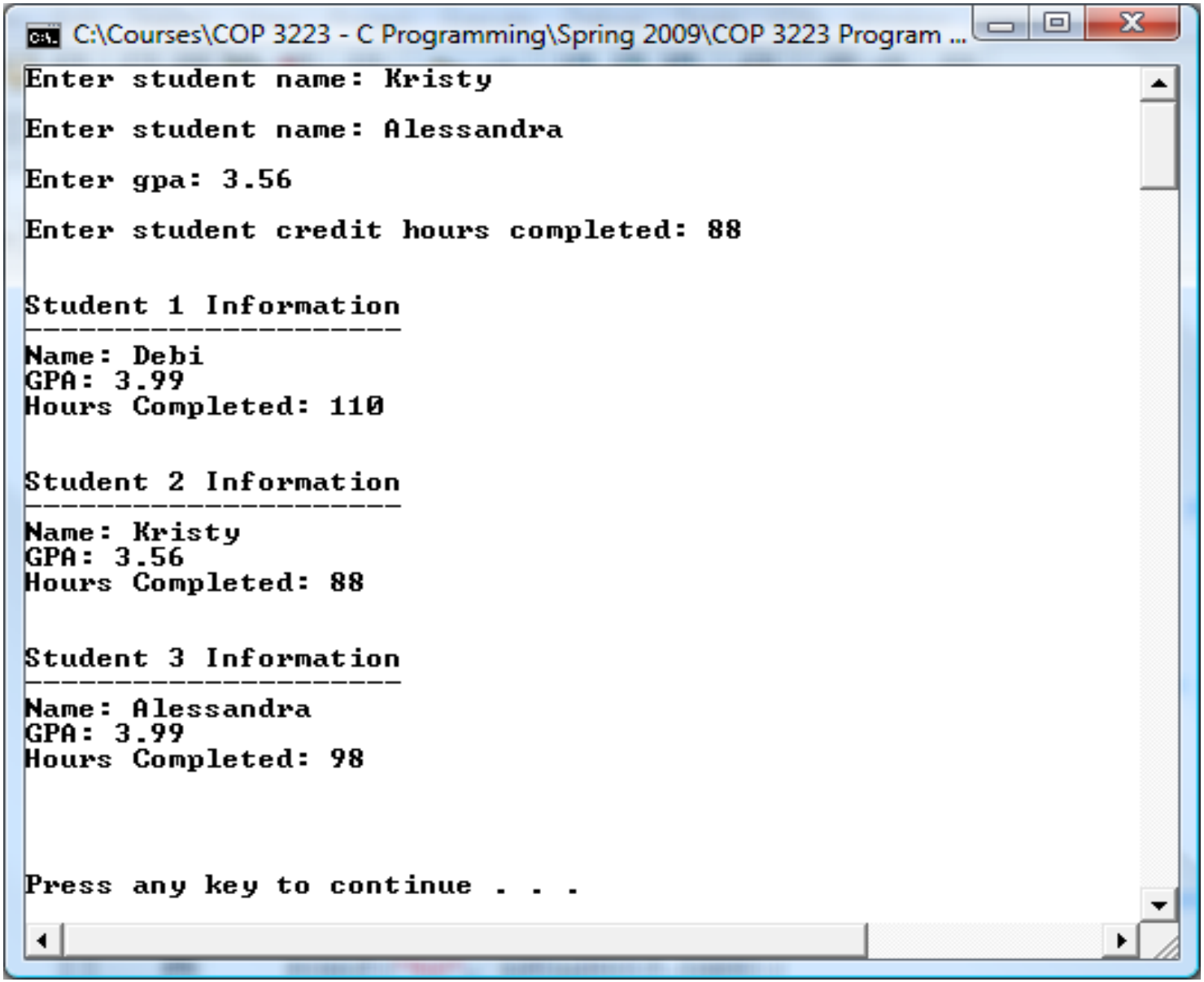
- Example: `student1.gpa`

Accessing Members Of Structures

- To illustrate some of the features of structures that we've seen so far, let's write a program that utilizes the `ucfStudent` structure we created on page 8.
- Notice the different way the structure members are assigned values using initializers, values read from the keyboard and direct assignment.

using structures - example 1.c

```
 6 #define MAXLENGTH 10
 7
 8 //structure defining a UCF student
 9 struct ucfStudent {
10    char name[MAXLENGTH]; //student's first name
11    double gpa; //student's gpa
12    int creditHoursCompleted; //hours completed by the student
13 } student1 = {"Debi", 3.99, 110};
14
15 int main()
16 {
17     struct ucfStudent student2, student3; //create two more students
18
19     printf("Enter student name: ");
20     scanf("%s", &student2.name);
21     printf("\n");
22     student2.gpa = 3.56; //assign student2 gpa
23     student2.creditHoursCompleted = 88; //assign student2 credit hours
24     printf("Enter student name: ");
25     scanf("%s", &student3.name);
26     printf("\nEnter gpa: ");
27     scanf("%f", &student3.gpa);
28     printf("\nEnter student credit hours completed: ");
29     scanf("%d", &student3.creditHoursCompleted);
30     printf("\n\n");
31     student3.gpa = student1.gpa;  //assigne student3 gpa same as student1
32     student3.creditHoursCompleted = student2.creditHoursCompleted + 10;
33
34     printf("Student 1 Information\n");
35     printf("---------------------\n");
36     printf("Name: %s\nGPA: %4.2f\nHours Completed: %d\n\n\n",
```



C:\Courses\COP 3223 - C Programming\Spring 2009\COP 3223 Program ...
Enter student name: Kristy
Enter student name: Alessandra
Enter gpa: 3.56
Enter student credit hours completed: 88
Student 1 Information
Name: Debi
GPA: 3.99
Hours Completed: 110
Student 2 Information
Name: Kristy
GPA: 3.56
Hours Completed: 88
Student 3 Information
Name: Alessandra
GPA: 3.99
Hours Completed: 98
Press any key to continue . . .

Accessing Members Of Structures

- Now let's look at the other operator used to access a structure, the structure pointer operator (->).
- This operator works when a pointer to a structure has been declared and we are working with the structure members through the pointer to the structure.
- As a running example, let's declare a structure that represents a normal playing card:

```
struct card {
        char *face[MAX];
        char *suit[MAX];
};
```

Accessing Members Of Structures

- Declaring a pointer to a `card` structure would be done as follows:

```
struct card *cardPtr;
```

- We can then access the members of the structure `card` using the structure pointer operator as follows:

```
cardPtr->face = "Ace";

cardPtr->suit = "Spades";
```

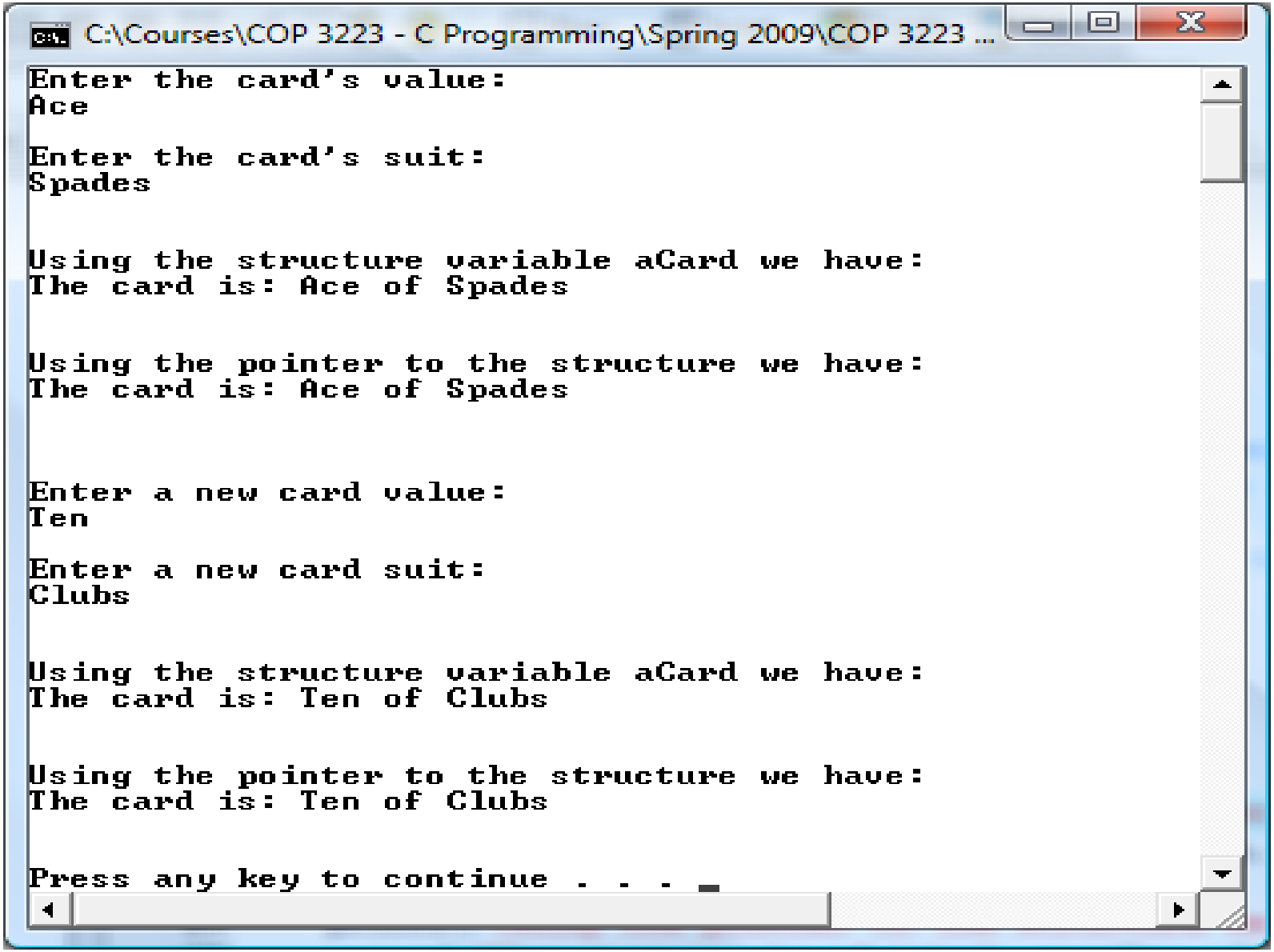
- Reading a value into a structure using a pointer is done in a similar manner , such as:

```
scanf("%s", &cardPtr->face);
```

- The following program illustrates the use of both the dot operator and the pointer structure operator.

using structures - example 1.c | [*] using structures - example 2.c

```
 5 #define MAX 10
 6
 7 struct card {
 8    char *face[MAX];
 9    char *suit[MAX];
10 };
11
12 int main ()
13 {
14     struct card aCard;  //define a single card structure
15     struct card *cardPtr;  //define a pointer to a card structure
16
17     cardPtr = &aCard; //assign address of aCard to cardPtr
18     printf("Enter the card's value: \n");
19     scanf("%s", &cardPtr->face);
20     //printf("%s\n", aCard.face);
21     printf("\nEnter the card's suit: \n");
22     scanf("%s", &cardPtr->suit);
23     printf("\n\nUsing the structure variable aCard we have:\n");
24     printf("The card is: %s of %s\n", aCard.face, aCard.suit);
25     printf("\n\nUsing the pointer to the structure we have:\n");
26     printf("The card is: %s of %s\n\n\n", cardPtr->face, cardPtr->suit);
27     //modifying the card's value
28     printf("Enter a new card value: \n");
29     scanf("%s", &cardPtr->face);
30     //printf("%s\n", aCard.face);
31     printf("\nEnter a new card suit: \n");
32     scanf("%s", &cardPtr->suit);
33     printf("\n\nUsing the structure variable aCard we have:\n");
34     printf("The card is: %s of %s\n", aCard.face, aCard.suit);
35     printf("\n\nUsing the pointer to the structure we have:\n");
36     printf("The card is: %s of %s\n", cardPtr->face, cardPtr->suit);
```



C:\Courses\COP 3223 - C Programming\Spring 2009\COP 3223 ...
Enter the card's value:
Ace
Enter the card's suit:
Spades
Using the structure variable aCard we have:
The card is: Ace of Spades
Using the pointer to the structure we have:
The card is: Ace of Spades
Enter a new card value:
Ten
Enter a new card suit:
Clubs
Using the structure variable aCard we have:
The card is: Ten of Clubs
Using the pointer to the structure we have:
The card is: Ten of Clubs
Press any key to continue . . .

The keyword `typedef`

- The keyword `typedef` provides a mechanism for creating aliases for previously defined data types. Names for structure types are often defined with `typedef` to create shorter type names.
- For example, the statement:

```
typedef struct card Card;
```

  defines a new type name `Card` as an alias for the type `struct card`.

- C convention is to use a capital letter for the type defined in a `typedef` statement.

The keyword `typedef`

- It is most common to use a typedef statement to define a type with a structure where the structure name (structure tag) is missing.
- For example, the card structure that we've been using would be defined as follows:

```
typedef struct {
        char *face[MAX];
        char *suit[MAX];
} Card;
```

- Notice that when using the typedef statement, that variables of the structure type cannot be declared between the closing } and the closing ;.

Using Structures With Functions

- Structures can be passed to functions by passing individual structure members, by passing an entire structure, or by passing a pointer to a structure.
- When structures or individual members of structures are passed to a function, they are passed by value. Therefore, it is impossible for the called function to modify members of the caller's structure.
- To pass a structure by reference you must pass the address of the structure variable to the function (i.e., a pointer to the structure). [In the following example, all the parameters passed to functions are arrays so implicit pointers are being passed in lines 77-79.]
- Arrays of structures (like any array) are passed by reference.

Using Structures With Functions

- The final example in this set of notes continues with the card structure, but introduces the `typedef` statement and uses several functions to which arrays of structures are passed.
- The program creates a deck of cards (an array of structures) and then uses functions to fill the deck with valid cards, shuffle the deck, and finally deal all the cards in the deck.

card shuffling example.c

```c
//Structures In C - Part 1 - Using structures with functions
//Example deals a deck of cards stored as a structure and uses functions
//to shuffle and deal the cards.
//April 14, 2009   Written by: Mark Llewellyn

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAXFACE 13
#define MAXCARDS 52

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

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

void fillDeck( Card * const wDeck, const char * wFace[], const char * wSuit[] )
{
   int i; // loop control

   // loop through wDeck
   for ( i = 0; i < MAXCARDS; i++ ) {
      wDeck[ i ].face = wFace[ i % MAXFACE ];
      wDeck[ i ].suit = wSuit[ i / MAXFACE ];
   }//end for stmt
}//end fillDeck function

```


card shuffling example.c

```
31 void shuffle( Card * const wDeck )
32 {
33    int i;      // loop control
34    int j;      ///holds random value between 0 - 51
35    Card temp; //defines temporary structure for swapping Cards
36
37    // loop through wDeck randomly swapping Cards
38    for ( i = 0; i < MAXCARDS; i++ ) {
39       j = rand() % MAXCARDS;
40       temp = wDeck[ i ];
41       wDeck[ i ] = wDeck[ j ];
42       wDeck[ j ] = temp;
43    }//end for stmt
44 }//end shuffle function
45
46 void deal( const Card * const wDeck )
47 {
48    int i; //loop control
49     printf("\nThe shuffled deck\n");
50     printf("-----------------\n\n");
51    // loop through wDeck
52    for ( i = 0; i < MAXCARDS; i++ ) {
53       printf( "%5s of %-8s%c", wDeck[ i ].face, wDeck[ i ].suit,
54          ( i + 1 ) % 3 ? '\t' : '\n' ); //puts cards into three columns
55    }//end for stmt
56    printf("\n\n");
57 } //end deal function
58
```


```
64
65 int main()
66 {
67    Card deck[MAXCARDS]; // define array of Cards - an array of structures
68
69    // initialize array of pointers to face value of card
70    const char *face[] = { "Ace", "Deuce", "Three", "Four", "Five",
71       "Six", "Seven", "Eight", "Nine", "Ten",
72       "Jack", "Queen", "King"};
73    // initialize array of pointers to card suit
74    const char *suit[] = { "Hearts", "Diamonds", "Clubs", "Spades"};
75
76    srand( time( NULL ) ); // randomize
77    fillDeck( deck, face, suit ); // load the deck with Cards
78    shuffle( deck ); //put Cards in random order
79    deal( deck ); // deal all 52 Cards
80    //do a second shuffle and deal to show the changes in the structures
81    //comment out the next two lines for only one shuffle and deal
82    shuffle(deck);
83    deal(deck);
84    system("PAUSE");
85    return 0;
86 } //end main function
87
```


K:\COP 3223 - Spring 2009\COP 3223 Program Files\Structures In C - Part 1\card shuffling...

```
The shuffled deck
-----------------

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


The shuffled deck
-----------------

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


Practice Problems

1. Write a C program that defines a structure for food that maintains the name of the food, a portion size of that food, and the number of calories in the portion size. Read the values into an array of food items from a file of data, then print the contents of the array of food similar to how we did it in the first example on page 12.