Computer Programming I

Ming-Feng Tsai (Victor Tsai)

Dept. of Computer Science
National Chengchi University

C Structures, Unions, Bit Manipulations and Enumerations

Objectives

- In this chapter, you'll learn
  - To create and use structures, unions and enumerations
  - To pass structures to functions by value and by reference
  - To manipulate data with the bitwise operations
  - To create bit field for storing data compactly

Introduction

- Structures—sometimes referred to as aggregates—are collections of related variables under one name.
- Structures may contain variables of many different data types—in contrast to arrays that contain only elements of the same data type.

Structure Definitions

- Structures are derived data types—they are constructed using objects of other types.
- Consider the following structure definition:

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

  - Keyword **`struct`** introduces the structure definition.
  - The identifier **`card`** is the structure tag, which names the structure definition and is used with the keyword **`struct`** to declare variables of the structure type.

Structure Definitions (Cont.)

- Variables declared within the braces of the structure definition are the structure's members.
- Each structure definition must end with a semicolon.
- The definition of **`struct card`** contains members **`face`** and **`suit`** of type **`char *`**.

Structure Definitions (Cont.)

Common Programming Error 10.1

Forgetting the semicolon that terminates a structure definition is a syntax error.

Structure Definitions (Cont.)

- Structure members can be of many types.
- For example

```
struct employee {
    char firstName[20];
    char lastName[20];
    int age;
    char gender;
    double hourlySalary;
};
```

Structure Definitions (Cont.)

- A structure **cannot** contain an instance of itself.
- For example,

```
struct employee2 {
    char firstName[20];
    char lastName[20];
    int age;
    char gender;
    double hourlySalary;
    struct employee2 person; /* ERROR */
    struct employee2 *ePtr; /* pointer */
};
```

Structure Definitions (Cont.)

- **`struct employee2 person;`**
  - contains an instance of itself (person), which is an error.
- **`struct employee2 *ePtr;`**
  - A structure containing a member that is a pointer to the same structure type is referred to as a self-referential structure.
  - Self-referential structures are used to build linked data structures

Structure Definitions (Cont.)

- Structure definitions **do not reserve any space** in memory; rather, each definition **creates a new data type** that is used to define variables.
- Structure variables are defined like variables of other types.
- The definition

```
struct card aCard, deck[52], *cardPtr;
```

Structure Definitions (Cont.)

- For example, the preceding definition could have been incorporated into the struct card structure definition as follows:

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

Structure Definitions (Cont.)

- The **structure tag name** is optional.

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

Structure Definitions (Cont.)

- The only valid operations that may be performed on structures are:
  - **assigning** structure variables to structure variables of the same type,
  - taking the **address (&)** of a structure variable,
  - **accessing the members** of a structure variable
  - using the **sizeof** operator to determine the size of a structure variable

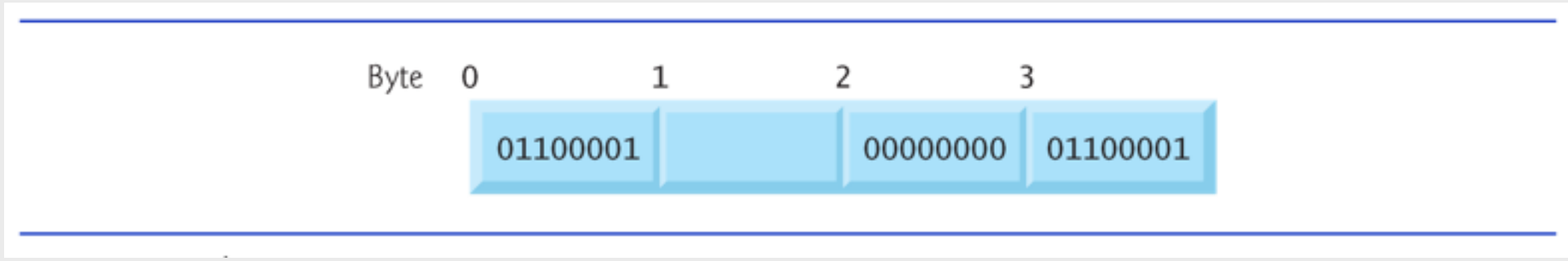
Structure Definitions (Cont.)

- **Structures may not be compared** using operators == and !=, because structure members are not necessarily stored in consecutive bytes of memory.

Structure Definitions (Cont.)

- Consider the following structure definition, in which **`sample1`** and **`sample2`** of type struct example are declared:

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



Initializing Structures

- To initialize a structure, follow the variable name in the definition with an equals sign and a brace-enclosed, comma-separated list of initializers.
- For example, the declaration

```
struct card { //define a new data type
   char *face;
   char *suit;
};

struct card aCard = {"Three","Hearts" };
// declaration and initialization
```

Accessing Structure Members

- Two operators are used to access members of structures: the structure member operator (`.`)—also called the dot operator—and the structure pointer operator (`->`)—also called the arrow operator.

- For example, to print member **`suit`** of structure variable **`aCard`**, use the statement

```
printf("%s", aCard.suit);
/* displays Hearts */
```

Accessing Structure Members (Cont.)

- The structure pointer operator—consisting of a minus (**–**) sign and a greater than (**>**) sign with **no intervening spaces**—accesses a structure member via a pointer to the structure.
- To print member **`suit`** of structure **`aCard`** with pointer **`cardPtr`**, use the statement

```
printf("%s", cardPtr->suit);
/* displays Hearts */
```

Accessing Structure Members (Cont.)

- The expression **`cardPtr->suit`** is equivalent to **`(*cardPtr).suit`**, which dereferences the pointer and accesses the member **`suit`** using the structure member operator.
- The parentheses are needed here because the structure member operator (`.`) has a higher precedence than the pointer dereferencing operator (`*`).

Accessing Structure Members (Cont.)

- Example: fig10_02.c

```
 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     struct card aCard; /* define one struct card variable */
14     struct card *cardPtr; /* define a pointer to a struct card */
15
16     /* place strings into aCard */
17     aCard.face = "Ace";
18     aCard.suit = "Spades";
19
20     cardPtr = &aCard; /* assign address of aCard to cardPtr */
21
22     printf( "%s%s%s\n%s%s%s\n%s%s%s\n", aCard.face, " of ", aCard.suit,
23             cardPtr->face, " of ", cardPtr->suit,
24             ( *cardPtr ).face, " of ", ( *cardPtr ).suit );
25     return 0; /* indicates successful termination */
26 } /* end main */
```

Accessing Structure Members (Cont.)

- Example: fig10_02.c

```
 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     struct card aCard; /* define one struct card variable */
14     struct card *cardPtr; /* define a pointer to a struct card */
15
16     /* place strings into aCard */
17     aCard.face = "Ace";
18     aCard.suit = "Spades";
19
20     cardPtr = &aCard; /* assign address of aCard to cardPtr */
21
22     printf( "%s%s%s\n%s%s%s\n%s%s%s\n", aCard.face, " of ", aCard.suit,
23             cardPtr->face, " of ", cardPtr->suit,
24             ( *cardPtr ).face, " of ", ( *cardPtr ).suit );
25     return 0; /* indicates successful termination */
26 } /* end main */
```

Accessing Structure Members (Cont.)

- Example: fig10_02.c

```c
struct card {
    char *face; /* define pointer face */
    char *suit; /* define pointer suit */
}; /* end structure card */

int main( void ) {
    struct card aCard; /* define one struct card variable */
    struct card *cardPtr; /* define a pointer to a struct card */

    /* place strings into aCard */
    aCard.face = "Ace";
    aCard.suit = "Spades";

    cardPtr = &aCard; /* assign address of aCard to cardPtr */

    printf( "%s%s%s\n%s%s%s\n%s%s%s\n", aCard.face, " of ", aCard.suit,
            cardPtr->face, " of ", cardPtr->suit,
            ( *cardPtr ).face, " of ", ( *cardPtr ).suit );
    return 0; /* indicates successful termination */
} /* end main */
```

define a new data type named **card**

Accessing Structure Members (Cont.)

- Example: fig10_02.c

```c
struct card {
    char *face; /* define pointer face */
    char *suit; /* define pointer suit */
}; /* end structure card */

int main( void ) {
    struct card aCard; /* define one struct card variable */
    struct card *cardPtr; /* define a pointer to a struct card */

    /* place strings into aCard */
    aCard.face = "Ace";
    aCard.suit = "Spades";

    cardPtr = &aCard; /* assign address of aCard to cardPtr */

    printf( "%s%s%s\n%s%s%s\n%s%s%s\n", aCard.face, " of ", aCard.suit,
            cardPtr->face, " of ", cardPtr->suit,
            ( *cardPtr ).face, " of ", ( *cardPtr ).suit );
    return 0; /* indicates successful termination */
} /* end main */
```

define a new data type named **card**

Accessing Structure Members (Cont.)

- Example: fig10_02.c

```c
struct card {
   char *face; /* define pointer face */
   char *suit; /* define pointer suit */
}; /* end structure card */

int main( void ) {
   struct card aCard; /* define one struct card variable */
   struct card *cardPtr; /* define a pointer to a struct card */

   /* place strings into aCard */
   aCard.face = "Ace";
   aCard.suit = "Spades";

   cardPtr = &aCard; /* assign address of aCard to cardPtr */

   printf( "%s%s%s\n%s%s%s\n%s%s%s\n", aCard.face, " of ", aCard.suit,
           cardPtr->face, " of ", cardPtr->suit,
           ( *cardPtr ).face, " of ", ( *cardPtr ).suit );
   return 0; /* indicates successful termination */
} /* end main */
```

define a new data type named **card**

use the new type to declare two variables: **aCard** and **cardPtr**

Accessing Structure Members (Cont.)

- Example: fig10_02.c

```c
struct card {
   char *face; /* define pointer face */
   char *suit; /* define pointer suit */
}; /* end structure card */

int main( void ) {
   struct card aCard; /* define one struct card variable */
   struct card *cardPtr; /* define a pointer to a struct card */

   /* place strings into aCard */
   aCard.face = "Ace";
   aCard.suit = "Spades";

   cardPtr = &aCard; /* assign address of aCard to cardPtr */

   printf( "%s%s%s\n%s%s%s\n%s%s%s\n", aCard.face, " of ", aCard.suit,
           cardPtr->face, " of ", cardPtr->suit,
           ( *cardPtr ).face, " of ", ( *cardPtr ).suit );
   return 0; /* indicates successful termination */
} /* end main */
```

define a new data type named **card**

use the new type to declare two variables: **aCard** and **cardPtr**

Accessing Structure Members (Cont.)

- Example: fig10_02.c

```
 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     struct card aCard; /* define one struct card variable */
14     struct card *cardPtr; /* define a pointer to a struct card */
15
16     /* place strings into aCard */
17     aCard.face = "Ace";
18     aCard.suit = "Spades";
19
20     cardPtr = &aCard; /* assign address of aCard to cardPtr */
21
22     printf( "%s%s%s\n%s%s%s\n%s%s%s\n", aCard.face, " of ", aCard.suit,
23             cardPtr->face, " of ", cardPtr->suit,
24             ( *cardPtr ).face, " of ", ( *cardPtr ).suit );
25     return 0; /* indicates successful termination */
26 } /* end main */
```

define a new data type named **card**

use the new type to declare two variables: **aCard** and **cardPtr**

use **dot** notation to access its members

Accessing Structure Members (Cont.)

- Example: fig10_02.c

```
 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     struct card aCard; /* define one struct card variable */
14     struct card *cardPtr; /* define a pointer to a struct card */
15
16     /* place strings into aCard */
17     aCard.face = "Ace";
18     aCard.suit = "Spades";
19
20     cardPtr = &aCard; /* assign address of aCard to cardPtr */
21
22     printf( "%s%s%s\n%s%s%s\n%s%s%s\n", aCard.face, " of ", aCard.suit,
23             cardPtr->face, " of ", cardPtr->suit,
24             ( *cardPtr ).face, " of ", ( *cardPtr ).suit );
25     return 0; /* indicates successful termination */
26 } /* end main */
```

define a new data type named **card**

use the new type to declare two variables: **aCard** and **cardPtr**

use **dot** notation to access its members

Accessing Structure Members (Cont.)

- Example: fig10_02.c

```c
struct card {
   char *face; /* define pointer face */
   char *suit; /* define pointer suit */
}; /* end structure card */

int main( void ) {
   struct card aCard; /* define one struct card variable */...
   struct card *cardPtr; /* define a pointer to a struct card */

   /* place strings into aCard */
   aCard.face = "Ace";
   aCard.suit = "Spades";

   cardPtr = &aCard; /* assign address of aCard to cardPtr */

   printf( "%s%s%s\n%s%s%s\n%s%s%s\n", aCard.face, " of ", aCard.suit,
           cardPtr->face, " of ", cardPtr->suit,
           ( *cardPtr ).face, " of ", ( *cardPtr ).suit );
   return 0; /* indicates successful termination */
} /* end main */
```

define a new data type named **card**

use the new type to declare two variables: **aCard** and **cardPtr**

use **dot** notation to access its members

use **Ptr->member** and **(*Ptr).member** to access its members

Accessing Structure Members (Cont.)

- Example: fig10_02.c

```c
struct card {
   char *face; /* define pointer face */
   char *suit; /* define pointer suit */
}; /* end structure card */

int main( void ) {
   struct card aCard; /* define one struct card variable */...
   struct card *cardPtr; /* define a pointer to a struct card */

   /* place strings into aCard */
   aCard.face = "Ace";
   aCard.suit = "Spades";

   cardPtr = &aCard; /* assign address of aCard to cardPtr */

   printf( "%s%s%s\n%s%s%s\n%s%s%s\n", aCard.face, " of ", aCard.suit,
           cardPtr->face, " of ", cardPtr->suit,
           ( *cardPtr ).face, " of ", ( *cardPtr ).suit );
   return 0; /* indicates successful termination */
} /* end main */
```

define a new data type named **card**

use the new type to declare two variables: **aCard** and **cardPtr**

use **dot** notation to access its members

use **Ptr->member** and **(*Ptr).member** to access its members

```
Ace of Spades
Ace of Spades
Ace of Spades
```

Using Structures with Functions

- When structures or individual structure members are passed to a function, they are **passed by value**.
  - Therefore, the members of a caller's structure cannot be modified by the called function.
- To pass a structure by reference, **pass the address** of the structure variable.

Using Structures with Functions (Cont.)

Common Programming Error 10.6

Assuming that structures, like arrays, are automatically passed by reference and trying to modify the caller's structure values in the called function is a logic error.

typedef

- The keyword **typedef** provides a mechanism for creating synonyms (or **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 the new type name **Card** as a synonym for type **struct card**.

`typedef` (Cont.)

- For example, the following definition

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

creates the structure type **`Card`** without the need for a separate **`typedef`** statement.

`typedef` (Cont.)

- Creating a new name with **`typedef`** does not create a new type; **`typedef`** simply creates a new type name, which may be used as an alias for an existing type name.

Example: High-Performance Card Shuffling and Dealing Simulation

- Based on the card shuffling and dealing simulation discussed in Chapter 7.
- The program represents the deck of cards as an array of structures.
- The program uses high-performance shuffling and dealing algorithms.

Example: High-Performance Card Shuffling and Dealing Simulation (Cont.)

- Example: fig10_03.c

```c
#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 */
```

Example: High-Performance Card Shuffling and Dealing Simulation (Cont.)

- Example: fig10_03.c

```c
#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 */
```

Example: High-Performance Card Shuffling and Dealing Simulation (Cont.)

- Example: fig10_03.c

```c
#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 */
```

card structure definition

Example: High-Performance Card Shuffling and Dealing Simulation (Cont.)

- Example: fig10_03.c

```c
#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 */
```

card structure definition

Example: High-Performance Card Shuffling and Dealing Simulation (Cont.)

- Example: fig10_03.c

```
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <time.h>
6
7  /* card structure definition */
8  struct card {
9      const char *face; /* define pointer face */
10     const char *suit; /* define pointer suit */
11 }; /* end structure card */
12
13 typedef struct card Card; /* new type name for struct card */
```

card structure definition

new type name for struct card

Example: High-Performance Card Shuffling and Dealing Simulation (Cont.)

- Example: fig10_03.c

```
20 int main( void ) {
21    Card deck[ 52 ]; /* define array of Cards */
22
23    /* initialize array of pointers */
24    const char *face[] = { "Ace", "Deuce", "Three", "Four", "Five",
25       "Six", "Seven", "Eight", "Nine", "Ten",
26       "Jack", "Queen", "King"};
27
28    /* initialize array of pointers */
29    const char *suit[] = { "Hearts", "Diamonds", "Clubs", "Spades"};
30
31    srand( time( NULL ) ); /* randomize */
32
33    fillDeck( deck, face, suit ); /* load the deck with Cards */
34    shuffle( deck ); /* put Cards in random order */
35    deal( deck ); /* deal all 52 Cards */
36    return 0; /* indicates successful termination */
37 } /* end main */
```

Example: High-Performance Card Shuffling and Dealing Simulation (Cont.)

- Example: fig10_03.c

```
20 int main( void ) {
21    Card deck[ 52 ]; /* define array of Cards */
22
23    /* initialize array of pointers */
24    const char *face[] = { "Ace", "Deuce", "Three", "Four", "Five",
25       "Six", "Seven", "Eight", "Nine", "Ten",
26       "Jack", "Queen", "King"};
27
28    /* initialize array of pointers */
29    const char *suit[] = { "Hearts", "Diamonds", "Clubs", "Spades"};
30
31    srand( time( NULL ) ); /* randomize */
32
33    fillDeck( deck, face, suit ); /* load the deck with Cards */
34    shuffle( deck ); /* put Cards in random order */
35    deal( deck ); /* deal all 52 Cards */
36    return 0; /* indicates successful termination */
37 } /* end main */
```

Example: High-Performance Card Shuffling and Dealing Simulation (Cont.)

- Example: fig10_03.c

```
20 int main( void ) {
21    Card deck[ 52 ]; /* define array of Cards */
22
23    /* initialize array of pointers */
24    const char *face[] = { "Ace", "Deuce", "Three", "Four", "Five",
25       "Six", "Seven", "Eight", "Nine", "Ten",
26       "Jack", "Queen", "King"};
27
28    /* initialize array of pointers */
29    const char *suit[] = { "Hearts", "Diamonds", "Clubs", "Spades"};
30
31    srand( time( NULL ) ); /* randomize */
32
33    fillDeck( deck, face, suit ); /* load the deck with Cards */
34    shuffle( deck ); /* put Cards in random order */
35    deal( deck ); /* deal all 52 Cards */
36    return 0; /* indicates successful termination */
37 } /* end main */
```

define array of Cards

Example: High-Performance Card Shuffling and Dealing Simulation (Cont.)

- Example: fig10_03.c

```
40 void fillDeck( Card * const wDeck, const char * wFace[],
41         const char * wSuit[] ) {
42     int i; /* counter */
43
44     /* loop through wDeck */
45     for ( i = 0; i <= 51; i++ ) {
46         wDeck[ i ].face = wFace[ i % 13 ];
47         wDeck[ i ].suit = wSuit[ i / 13 ];
48     } /* end for */
49 } /* end function fillDeck */
```

Example: High-Performance Card Shuffling and Dealing Simulation (Cont.)

- Example: fig10_03.c

```
40 void fillDeck( Card * const wDeck, const char * wFace[],
41         const char * wSuit[] ) {
42     int i; /* counter */
43
44     /* loop through wDeck */
45     for ( i = 0; i <= 51; i++ ) {
46         wDeck[ i ].face = wFace[ i % 13 ];
47         wDeck[ i ].suit = wSuit[ i / 13 ];
48     } /* end for */
49 } /* end function fillDeck */
```

Example: High-Performance Card Shuffling and Dealing Simulation (Cont.)

- Example: fig10_03.c

```
40 void fillDeck( Card * const wDeck, const char * wFace[],
41         const char * wSuit[] ) {
42     int i; /* counter */
43
44     /* loop through wDeck */
45     for ( i = 0; i <= 51; i++ ) {
46         wDeck[ i ].face = wFace[ i % 13 ];
47         wDeck[ i ].suit = wSuit[ i / 13 ];
48     } /* end for */
49 } /* end function fillDeck */
```

receive an array of Cards

Example: High-Performance Card Shuffling and Dealing Simulation (Cont.)

- Example: fig10_03.c

```
40 void fillDeck( Card * const wDeck, const char * wFace[],
41         const char * wSuit[] ) {
42    int i; /* counter */
43
44    /* loop through wDeck */
45    for ( i = 0; i <= 51; i++ ) {
46       wDeck[ i ].face = wFace[ i % 13 ];
47       wDeck[ i ].suit = wSuit[ i / 13 ];
48    } /* end for */
49 } /* end function fillDeck */
```

receive an array of Cards

Example: High-Performance Card Shuffling and Dealing Simulation (Cont.)

- Example: fig10_03.c

```
40 void fillDeck( Card * const wDeck, const char * wFace[],
41         const char * wSuit[] ) {
42    int i; /* counter */
43
44    /* loop through wDeck */
45    for ( i = 0; i <= 51; i++ ) {
46       wDeck[ i ].face = wFace[ i % 13 ];
47       wDeck[ i ].suit = wSuit[ i / 13 ];
48    } /* end for */
49 } /* end function fillDeck */
```

receive an array of Cards

loop through wDeck

Example: High-Performance Card Shuffling and Dealing Simulation (Cont.)

- Example: fig10_03.c

```
52 void shuffle( Card * const wDeck ) {
53     int i; /* counter */
54     int j; /* variable to hold random value between 0 - 51 */
55     Card temp; /* define temporary structure for swapping Cards */
56
57     /* loop through wDeck randomly swapping Cards */
58     for ( i = 0; i <= 51; i++ ) {
59         j = rand() % 52;
60         temp = wDeck[ i ];
61         wDeck[ i ] = wDeck[ j ];
62         wDeck[ j ] = temp;
63     } /* end for */
64 } /* end function shuffle */
```

Example: High-Performance Card Shuffling and Dealing Simulation (Cont.)

- Example: fig10_03.c

```
52 void shuffle( Card * const wDeck ) {
53     int i; /* counter */
54     int j; /* variable to hold random value between 0 - 51 */
55     Card temp; /* define temporary structure for swapping Cards */
56
57     /* loop through wDeck randomly swapping Cards */
58     for ( i = 0; i <= 51; i++ ) {
59         j = rand() % 52;
60         temp = wDeck[ i ];
61         wDeck[ i ] = wDeck[ j ];
62         wDeck[ j ] = temp;
63     } /* end for */
64 } /* end function shuffle */
```

Example: High-Performance Card Shuffling and Dealing Simulation (Cont.)

- Example: fig10_03.c

```
52 void shuffle( Card * const wDeck ) {
53     int i; /* counter */
54     int j; /* variable to hold random value between 0 - 51 */
55     Card temp; /* define temporary structure for swapping Cards */
56
57     /* loop through wDeck randomly swapping Cards */
58     for ( i = 0; i <= 51; i++ ) {
59         j = rand() % 52;
60         temp = wDeck[ i ];
61         wDeck[ i ] = wDeck[ j ];
62         wDeck[ j ] = temp;
63     } /* end for */
64 } /* end function shuffle */
```

Example: High-Performance Card Shuffling and Dealing Simulation (Cont.)

- Example: fig10_03.c

```
52 void shuffle( Card * const wDeck ) {
53     int i; /* counter */
54     int j; /* variable to hold random value between 0 - 51 */
55     Card temp; /* define temporary structure for swapping Cards */
56
57     /* loop through wDeck randomly swapping Cards */
58     for ( i = 0; i <= 51; i++ ) {
59         j = rand() % 52;
60         temp = wDeck[ i ];
61         wDeck[ i ] = wDeck[ j ];
62         wDeck[ j ] = temp;
63     } /* end for */
64 } /* end function shuffle */
```

randomly swapping Cards; a total of 52 swaps are made in a single pass of the entire array

Example: High-Performance Card Shuffling and Dealing Simulation (Cont.)

- Example: fig10_03.c

```
67 void deal( const Card * const wDeck ) {
68    int i; /* counter */
69
70    /* loop through wDeck */
71    for ( i = 0; i <= 51; i++ ) {
72       printf( "%5s of %-8s%s", wDeck[ i ].face, wDeck[ i ].suit,
73               ( i + 1 ) % 4 ? "  " : "\n" );
74    } /* end for */
75 } /* end function deal */
```

Example: High-Performance Card Shuffling and Dealing Simulation (Cont.)

- Example: fig10_03.c

```
67 void deal( const Card * const wDeck ) {
68    int i; /* counter */
69
70    /* loop through wDeck */
71    for ( i = 0; i <= 51; i++ ) {
72       printf( "%5s of %-8s%s", wDeck[ i ].face, wDeck[ i ].suit,
73               ( i + 1 ) % 4 ? "  " : "\n" );
74    } /* end for */
75 } /* end function deal */
```

Example: High-Performance Card Shuffling and Dealing Simulation (Cont.)

- Example: fig10_03.c

```
67 void deal( const Card * const wDeck ) {
68    int i; /* counter */
69
70    /* loop through wDeck */
71    for ( i = 0; i <= 51; i++ ) {
72       printf( "%5s of %-8s%s", wDeck[ i ].face, wDeck[ i ].suit,
73               ( i + 1 ) % 4 ? "  " : "\n" );
74    } /* end for */
75 } /* end function deal */
```

Example: High-Performance Card Shuffling and Dealing Simulation (Cont.)

- Example: fig10_03.c

```
67 void deal( const Card * const wDeck ) {
68    int i; /* counter */
69
70    /* loop through wDeck */
71    for ( i = 0; i <= 51; i++ ) {
72       printf( "%5s of %-8s%s", wDeck[ i ].face, wDeck[ i ].suit,
73             ( i + 1 ) % 4 ? "  " : "\n" );
74    } /* end for */
75 } /* end function deal */
```

print the card face and suit

Example: High-Performance Card Shuffling and Dealing Simulation (Cont.)

- Example: fig10_03.c

```
67 void deal( const Card * const wDeck ) {
68    int i; /* counter */
69
70    /* loop through wDeck */
71    for ( i = 0; i <= 51; i++ ) {
72       printf( "%5s of %-8s%s", wDeck[ i ].face, wDeck[ i ].suit,
73             ( i + 1 ) % 4 ? "  " : "\n" );
74    } /* end for */
75 } /* end function deal */
```

print the card face and suit

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

Example: High-Performance Card Shuffling and Dealing Simulation (Cont.)

Common Programming Error 10.7

Forgetting to include the array subscript when referring to individual structures in an array of structures is a syntax error.

Unions

- A **`union`** is a derived data type—like a structure—with members that share the same storage space.
- For different situations in a program, some variables may not be relevant, but other variables are—so a union shares the space instead of wasting storage on variables that are not being used.
- The members of a union can be of any data type.

Unions (Cont.)

- A union is declared with keyword **`union`** in the same format as a structure.
- The union definition

```
union number {
   int x;
   double y;
};
```

  indicates that **`number`** is a **`union`** type with members **`int x`** and **`double y`**.

- The **`union`** definition is normally placed in a header and included in all source files that use the union type.

Unions (Cont.)

- The operations that can be performed on a union are the following:
  - assigning (=) a union to another union of the same type,
  - taking the address (**&**) of a union variable,
  - and accessing union members using the structure member operator and the structure pointer operator.
- Unions may not be compared using operators **==** and **!=** for the same reasons that structures cannot be compared.

Unions (Cont.)

- In a declaration, a union may be initialized with a value of the same type as the first union member.
- For example, with the preceding union, the declaration

```
union number value = {10};
```

is a valid initialization of union variable **`value`**

Unions (Cont.)

- Example: fig10_05.c

```c
union number {
    int x;
    double y;
}; /* end union number */

int main( void ) {
    union number value; /* define union variable */

    value.x = 100; /* put an integer into the union */
    printf( "%s\n%s\n%s\n  %d\n\n%s\n  %f\n\n\n",
            "Put a value in the integer member",
            "and print both members.",
            "int:", value.x,
            "double:", value.y );

    value.y = 100.0; /* put a double into the same union */
    printf( "%s\n%s\n%s\n  %d\n\n%s\n  %f\n",
            "Put a value in the floating member",
            "and print both members.",
            "int:", value.x,
            "double:", value.y );
```

Unions (Cont.)

- Example: fig10_05.c

```c
union number {
    int x;
    double y;
}; /* end union number */

int main( void ) {
    union number value; /* define union variable */

    value.x = 100; /* put an integer into the union */
    printf( "%s\n%s\n%s\n  %d\n\n%s\n  %f\n\n\n",
            "Put a value in the integer member",
            "and print both members.",
            "int:", value.x,
            "double:", value.y );

    value.y = 100.0; /* put a double into the same union */
    printf( "%s\n%s\n%s\n  %d\n\n%s\n  %f\n",
            "Put a value in the floating member",
            "and print both members.",
            "int:", value.x,
            "double:", value.y );
```

Unions (Cont.)

- Example: fig10_05.c

```c
union number {
    int x;
    double y;
}; /* end union number */

int main( void ) {
    union number value; /* define union variable */

    value.x = 100; /* put an integer into the union */
    printf( "%s\n%s\n%s\n  %d\n\n%s\n  %f\n\n\n",
            "Put a value in the integer member",
            "and print both members.",
            "int:", value.x,
            "double:", value.y );

    value.y = 100.0; /* put a double into the same union */
    printf( "%s\n%s\n%s\n  %d\n\n%s\n  %f\n",
            "Put a value in the floating member",
            "and print both members.",
            "int:", value.x,
            "double:", value.y );
```

use union to define a new type named **number**

Unions (Cont.)

- Example: fig10_05.c

```c
union number {
    int x;
    double y;
}; /* end union number */

int main( void ) {
    union number value; /* define union variable */

    value.x = 100; /* put an integer into the union */
    printf( "%s\n%s\n%s\n  %d\n\n%s\n  %f\n\n\n",
            "Put a value in the integer member",
            "and print both members.",
            "int:", value.x,
            "double:", value.y );

    value.y = 100.0; /* put a double into the same union */
    printf( "%s\n%s\n%s\n  %d\n\n%s\n  %f\n",
            "Put a value in the floating member",
            "and print both members.",
            "int:", value.x,
            "double:", value.y );
```

use union to define a new type named **number**

Unions (Cont.)

- Example: fig10_05.c

```c
union number {
    int x;
    double y;
}; /* end union number */

int main( void ) {
    union number value; /* define union variable */

    value.x = 100; /* put an integer into the union */
    printf( "%s\n%s\n%s\n  %d\n\n%s\n  %f\n\n\n",
            "Put a value in the integer member",
            "and print both members.",
            "int:", value.x,
            "double:", value.y );

    value.y = 100.0; /* put a double into the same union */
    printf( "%s\n%s\n%s\n  %d\n\n%s\n  %f\n",
            "Put a value in the floating member",
            "and print both members.",
            "int:", value.x,
            "double:", value.y );
```

use union to define a new type named **number**

define an union variable named **value**

Unions (Cont.)

- Example: fig10_05.c

```c
union number {
    int x;
    double y;
}; /* end union number */

int main( void ) {
    union number value; /* define union variable */

    value.x = 100; /* put an integer into the union */
    printf( "%s\n%s\n%s\n  %d\n\n%s\n  %f\n\n\n",
            "Put a value in the integer member",
            "and print both members.",
            "int:", value.x,
            "double:", value.y );

    value.y = 100.0; /* put a double into the same union */
    printf( "%s\n%s\n%s\n  %d\n\n%s\n  %f\n",
            "Put a value in the floating member",
            "and print both members.",
            "int:", value.x,
            "double:", value.y );
```

use union to define a new type named **number**

define an union variable named **value**

Unions (Cont.)

- Example: fig10_05.c

```
 6 union number {
 7     int x;
 8     double y;
 9 }; /* end union number */
10
11 int main( void ) {
12     union number value; /* define union variable */
13
14     value.x = 100; /* put an integer into the union */
15     printf( "%s\n%s\n%s\n  %d\n\n%s\n  %f\n\n\n",
16             "Put a value in the integer member",
17             "and print both members.",
18             "int:", value.x,
19             "double:", value.y );
20
21     value.y = 100.0; /* put a double into the same union */
22     printf( "%s\n%s\n%s\n  %d\n\n%s\n  %f\n",
23             "Put a value in the floating member",
24             "and print both members.",
25             "int:", value.x,
26             "double:", value.y );
```

use union to define a new type named **number**

define an union variable named **value**

put an integer into the union

Unions (Cont.)

- Example: fig10_05.c

```
 6 union number {
 7     int x;
 8     double y;
 9 }; /* end union number */
10
11 int main( void ) {
12     union number value; /* define union variable */
13
14     value.x = 100; /* put an integer into the union */
15     printf( "%s\n%s\n%s\n  %d\n\n%s\n  %f\n\n\n",
16             "Put a value in the integer member",
17             "and print both members.",
18             "int:", value.x,
19             "double:", value.y );
20
21     value.y = 100.0; /* put a double into the same union */
22     printf( "%s\n%s\n%s\n  %d\n\n%s\n  %f\n",
23             "Put a value in the floating member",
24             "and print both members.",
25             "int:", value.x,
26             "double:", value.y );
```

use union to define a new type named **number**

define an union variable named **value**

put an integer into the union

Unions (Cont.)

- Example: fig10_05.c

```
 6 union number {
 7     int x;
 8     double y;
 9 }; /* end union number */
10
11 int main( void ) {
12     union number value; /* define union variable */
13
14     value.x = 100; /* put an integer into the union */
15     printf( "%s\n%s\n%s\n  %d\n\n%s\n  %f\n\n\n",
16             "Put a value in the integer member",
17             "and print both members.",
18             "int:", value.x,
19             "double:", value.y );
20
21     value.y = 100.0; /* put a double into the same union */
22     printf( "%s\n%s\n%s\n  %d\n\n%s\n  %f\n",
23             "Put a value in the floating member",
24             "and print both members.",
25             "int:", value.x,
26             "double:", value.y );
```

use union to define a new type named **number**

define an union variable named **value**

put an integer into the union

put a double into the union

Unions (Cont.)

- Example: fig10_05.c

```
28    printf( "\n\n%s\n%s\n  %p\n\n%s\n  %p\n",
29            "Output the addresses of the two members:",
30            "x:", &(value.x),
31            "y:", &(value.y));
```

Unions (Cont.)

- Example: fig10_05.c

```
28    printf( "\n\n%s\n%s\n  %p\n\n%s\n  %p\n",
29            "Output the addresses of the two members:",
30            "x:", &(value.x),
31            "y:", &(value.y));
```

Unions (Cont.)

- Example: fig10_05.c

```
28    printf( "\n\n%s\n%s\n  %p\n\n%s\n  %p\n",
29            "Output the addresses of the two members:",
30            "x:", &(value.x),
31            "y:", &(value.y));
```

two members are at the same address

Unions (Cont.)

- Example: fig10_05.c

```
28    printf( "\n\n%s\n%s\n  %p\n\n%s\n  %p\n",
29            "Output the addresses of the two members:",
30            "x:", &(value.x),
31            "y:", &(value.y));
```

two members are at the same address

```
Put a value in the integer member
and print both members.
int:
  100

double:
  0.000000
```

Unions (Cont.)

- Example: fig10_05.c

```
28    printf( "\n\n%s\n%s\n  %p\n\n%s\n  %p\n",
29            "Output the addresses of the two members:",
30            "x:", &(value.x),
31            "y:", &(value.y));
```

two members are at the same address

```
Put a value in the integer member
and print both members.
int:
  100

double:
  0.000000
```

```
Put a value in the floating member
and print both members.
int:
  0

double:
  100.000000
```

Unions (Cont.)

- Example: fig10_05.c

```
28    printf( "\n\n%s\n%s\n  %p\n\n%s\n  %p\n",
29            "Output the addresses of the two members:",
30            "x:", &(value.x),
31            "y:", &(value.y));
```

two members are at the same address

```
Put a value in the integer member
and print both members.
int:
  100

double:
  0.000000
```

```
Put a value in the floating member
and print both members.
int:
  0

double:
  100.000000
```

```
Output the addresses of the two members:
x:
  0x7fff5fbfed90

y:
  0x7fff5fbfed90
```

Unions (Cont.)

- Example: fig10_06.c

```c
union udata {
    int x;
    long y;
    double z;
    char *a;
};

struct sdata {
    int x;
    long y;
    double z;
    char *a;
};

int main(void) {
    printf("%ld\n", sizeof(union udata));
    printf("%ld\n", sizeof(struct sdata));
    return 0;
}
```

Unions (Cont.)

- Example: fig10_06.c

```c
union udata {
    int x;
    long y;
    double z;
    char *a;
};

struct sdata {
    int x;
    long y;
    double z;
    char *a;
};

int main(void) {
    printf("%ld\n", sizeof(union udata));
    printf("%ld\n", sizeof(struct sdata));
    return 0;
}
```

Unions (Cont.)

- Example: fig10_06.c

```c
union udata {
    int x;
    long y;
    double z;
    char *a;
};

struct sdata {
    int x;
    long y;
    double z;
    char *a;
};

int main(void) {
    printf("%ld\n", sizeof(union udata));
    printf("%ld\n", sizeof(struct sdata));
    return 0;
}
```

define a union with four members

Unions (Cont.)

- Example: fig10_06.c

```c
union udata {
    int x;
    long y;
    double z;
    char *a;
};

struct sdata {
    int x;
    long y;
    double z;
    char *a;
};

int main(void) {
    printf("%ld\n", sizeof(union udata));
    printf("%ld\n", sizeof(struct sdata));
    return 0;
}
```

define a union with four members

Unions (Cont.)

- Example: fig10_06.c

```c
union udata {
    int x;
    long y;
    double z;
    char *a;
};

struct sdata {
    int x;
    long y;
    double z;
    char *a;
};

int main(void) {
    printf("%ld\n", sizeof(union udata));
    printf("%ld\n", sizeof(struct sdata));
    return 0;
}
```

define a union with four members

define a struct with the same four members

Unions (Cont.)

- Example: fig10_06.c

```c
union udata {
    int x;
    long y;
    double z;
    char *a;
};

struct sdata {
    int x;
    long y;
    double z;
    char *a;
};

int main(void) {
    printf("%ld\n", sizeof(union udata));
    printf("%ld\n", sizeof(struct sdata));
    return 0;
}
```

define a union with four members

define a struct with the same four members

Unions (Cont.)

- Example: fig10_06.c

```c
union udata {
    int x;
    long y;
    double z;
    char *a;
};

struct sdata {
    int x;
    long y;
    double z;
    char *a;
};

int main(void) {
    printf("%ld\n", sizeof(union udata));
    printf("%ld\n", sizeof(struct sdata));
    return 0;
}
```

define a union with four members

define a struct with the same four members

output the sizes of the above data types

Unions (Cont.)

- Example: fig10_06.c

```c
union udata {
    int x;
    long y;
    double z;
    char *a;
};

struct sdata {
    int x;
    long y;
    double z;
    char *a;
};

int main(void) {
    printf("%ld\n", sizeof(union udata));
    printf("%ld\n", sizeof(struct sdata));
    return 0;
}
```

define a union with four members

define a struct with the same four members

output the sizes of the above data types

```
8
32
```

Unions (Cont.)

- Example: fig10_06.c

```c
union udata {
    int x;
    long y;
    double z;
    char *a;
};

struct sdata {
    int x;
    long y;
    double z;
    char *a;
};

int main(void) {
    printf("%ld\n", sizeof(union udata));
    printf("%ld\n", sizeof(struct sdata));
    return 0;
}
```

define a union with four members

define a struct with the same four members

output the sizes of the above data types

```
8
32
```

because of 64-bits; in 32-bits, it will be 20.

Enumeration Constants

- C provides one final user-defined type called an enumeration.
- An enumeration, introduced by the keyword enum, is a set of integer enumeration constants represented by identifiers.
- For example, the enumeration

```
enum months {
   JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,
   OCT,NOV,DEC};
```

creates a new type, **`enum months`**, in which the identifiers are set to the integers **`0`** to **`11`**, respectively.

Enumeration Constants (Cont.)

- Values in an **`enum`** start with **`0`**, unless specified otherwise, and are incremented by **`1`**.
- To number the months 1 to 12, use the following enumeration:

```
enum months {
    JAN=1,FEB,MAR,APR,MAY,JUN,JUL,AUG,
    SEP,OCT,NOV,DEC};
```

- The identifiers in an enumeration must be unique.

Enumeration Constants (Cont.)

- Example: fig10_18.c

```c
enum months { JAN = 1, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC };

int main( void ) {
    enum months month; /* can contain any of the 12 months */

    /* initialize array of pointers */
    const char *monthName[] = { "", "January", "February", "March",
        "April", "May", "June", "July", "August", "September", "October",
        "November", "December" };

    /* loop through months */
    for ( month = JAN; month <= DEC; month++ ) {
        printf( "%2d%11s\n", month, monthName[ month ] );
    } /* end for */

    return 0; /* indicates successful termination */
} /* end main */
```

Enumeration Constants (Cont.)

- Example: fig10_18.c

```
6  enum months { JAN = 1, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC };
7
8  int main( void ) {
9      enum months month; /* can contain any of the 12 months */
10
11     /* initialize array of pointers */
12     const char *monthName[] = { "", "January", "February", "March",
13         "April", "May", "June", "July", "August", "September", "October",
14         "November", "December" };
15
16     /* loop through months */
17     for ( month = JAN; month <= DEC; month++ ) {
18         printf( "%2d%11s\n", month, monthName[ month ] );
19     } /* end for */
20
21     return 0; /* indicates successful termination */
22 } /* end main */
```

Enumeration Constants (Cont.)

- Example: fig10_18.c

```c
enum months { JAN = 1, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC };

int main( void ) {
    enum months month; /* can contain any of the 12 months */

    /* initialize array of pointers */
    const char *monthName[] = { "", "January", "February", "March",
        "April", "May", "June", "July", "August", "September", "October",
        "November", "December" };

    /* loop through months */
    for ( month = JAN; month <= DEC; month++ ) {
        printf( "%2d%11s\n", month, monthName[ month ] );
    } /* end for */

    return 0; /* indicates successful termination */
} /* end main */
```

create enumeration constants

Enumeration Constants (Cont.)

- Example: fig10_18.c

```c
enum months { JAN = 1, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC };

int main( void ) {
    enum months month; /* can contain any of the 12 months */

    /* initialize array of pointers */
    const char *monthName[] = { "", "January", "February", "March",
        "April", "May", "June", "July", "August", "September", "October",
        "November", "December" };

    /* loop through months */
    for ( month = JAN; month <= DEC; month++ ) {
        printf( "%2d%11s\n", month, monthName[ month ] );
    } /* end for */

    return 0; /* indicates successful termination */
} /* end main */
```

create enumeration constants

Enumeration Constants (Cont.)

- Example: fig10_18.c

```c
enum months { JAN = 1, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC };

int main( void ) {
    enum months month; /* can contain any of the 12 months */

    /* initialize array of pointers */
    const char *monthName[] = { "", "January", "February", "March",
        "April", "May", "June", "July", "August", "September", "October",
        "November", "December" };

    /* loop through months */
    for ( month = JAN; month <= DEC; month++ ) {
        printf( "%2d%11s\n", month, monthName[ month ] );
    } /* end for */

    return 0; /* indicates successful termination */
} /* end main */
```

create enumeration constants

month contain any of the 12 months

Enumeration Constants (Cont.)

- Example: fig10_18.c

```c
enum months { JAN = 1, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC };

int main( void ) {
    enum months month; /* can contain any of the 12 months */

    /* initialize array of pointers */
    const char *monthName[] = { "", "January", "February", "March",
        "April", "May", "June", "July", "August", "September", "October",
        "November", "December" };

    /* loop through months */
    for ( month = JAN; month <= DEC; month++ ) {
        printf( "%2d%11s\n", month, monthName[ month ] );
    } /* end for */

    return 0; /* indicates successful termination */
} /* end main */
```

create enumeration constants

month contain any of the 12 months

Enumeration Constants (Cont.)

- Example: fig10_18.c

```
6  enum months { JAN = 1, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC };
7
8  int main( void ) {
9     enum months month; /* can contain any of the 12 months */
10
11    /* initialize array of pointers */
12    const char *monthName[] = { "", "January", "February", "March",
13       "April", "May", "June", "July", "August", "September", "October",
14       "November", "December" };
15
16    /* loop through months */
17    for ( month = JAN; month <= DEC; month++ ) {
18       printf( "%2d%11s\n", month, monthName[ month ] );
19    } /* end for */
20
21    return 0; /* indicates successful termination */
22 } /* end main */
```

create enumeration constants

month contain any of the 12 months

loop through months

Enumeration Constants (Cont.)

- Example: fig10_18.c

```c
enum months { JAN = 1, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC };

int main( void ) {
    enum months month; /* can contain any of the 12 months */

    /* initialize array of pointers */
    const char *monthName[] = { "", "January", "February", "March",
        "April", "May", "June", "July", "August", "September", "October",
        "November", "December" };

    /* loop through months */
    for ( month = JAN; month <= DEC; month++ ) {
        printf( "%2d%11s\n", month, monthName[ month ] );
    } /* end for */

    return 0; /* indicates successful termination */
} /* end main */
```

create enumeration constants

month contain any of the 12 months

loop through months

```
 1    January
 2   February
 3      March
 4      April
 5        May
 6       June
 7       July
 8     August
 9  September
```

Bitwise Operators

- Each bit can assume the value 0 or the value 1.
- On most systems, a sequence of 8 bits forms a byte—the standard storage unit for a variable of type **`char`**.
- The bitwise operators are used to manipulate the bits of integral operands (**`char`**, **`short`**, **`int`** and **`long`**; both **`signed`** and **`unsigned`**).

Bitwise Operators (Cont.)

- For a detailed explanation of the binary (also called base-2) number system see Appendix C.
- Because of the machine-dependent nature of bitwise manipulations, these programs may not work on your system.
- The bitwise operators are bitwise AND (**&**), bitwise inclusive OR (**|**), bitwise exclusive OR (**^**), left shift (**<<**), right shift (**>>**) and complement (**~**).

Bitwise Operators (Cont.)

Operator		Description
&	bitwise AND	The bits in the result are set to 1 if the corresponding bits in the two operands are both 1.
|	bitwise inclusive OR	The bits in the result are set to 1 if at least one of the corresponding bits in the two operands is 1.
^	bitwise exclusive OR	The bits in the result are set to 1 if exactly one of the corresponding bits in the two operands is 1.
<<	left shift	Shifts the bits of the first operand left by the number of bits specified by the second operand; fill from the right with 0 bits.
>>	right shift	Shifts the bits of the first operand right by the number of bits specified by the second operand; the method of filling from the left is machine dependent.
~	one's complement	All 0 bits are set to 1 and all 1 bits are set to 0.

Bitwise Operators (Cont.)

- Example: fig10_07.c

```
 5 void displayBits( unsigned value ); /* prototype */
 6
 7 int main( void ) {
 8    unsigned x; /* variable to hold user input */
 9
10    printf( "Enter an unsigned integer: " );
11    scanf( "%u", &x );
12
13    displayBits( x );
14    return 0; /* indicates successful termination */
15 } /* end main */
```

Bitwise Operators (Cont.)

- Example: fig10_07.c

```
 5 void displayBits( unsigned value ); /* prototype */
 6
 7 int main( void ) {
 8    unsigned x; /* variable to hold user input */
 9
10    printf( "Enter an unsigned integer: " );
11    scanf( "%u", &x );
12
13    displayBits( x );
14    return 0; /* indicates successful termination */
15 } /* end main */
```

Bitwise Operators (Cont.)

- Example: fig10_07.c

```
 5 void displayBits( unsigned value ); /* prototype */
 6
 7 int main( void ) {
 8    unsigned x; /* variable to hold user input */
 9
10    printf( "Enter an unsigned integer: " );
11    scanf( "%u", &x );
12
13    displayBits( x );
14    return 0; /* indicates successful termination */
15 } /* end main */
```

enter an unsigned integer

Bitwise Operators (Cont.)

- Example: fig10_07.c

```c
void displayBits( unsigned value ) {
    unsigned c; /* counter */

    /* define displayMask and left shift 31 bits */
    unsigned displayMask = 1 << 31;

    printf( "%10u = ", value );

    /* loop through bits */
    for ( c = 1; c <= 32; c++ ) {
        putchar( value & displayMask ? '1' : '0' );
        value <<= 1; /* shift value left by 1 */

        if ( c % 8 == 0 ) { /* output space after 8 bits */
            putchar( ' ' );
        } /* end if */
    } /* end for */

    putchar( '\n' );
} /* end function displayBits */
```

Bitwise Operators (Cont.)

- Example: fig10_07.c

```
18 void displayBits( unsigned value ) {
19     unsigned c; /* counter */
20
21     /* define displayMask and left shift 31 bits */
22     unsigned displayMask = 1 << 31;
23
24     printf( "%10u = ", value );
25
26     /* loop through bits */
27     for ( c = 1; c <= 32; c++ ) {
28         putchar( value & displayMask ? '1' : '0' );
29         value <<= 1; /* shift value left by 1 */
30
31         if ( c % 8 == 0 ) { /* output space after 8 bits */
32             putchar( ' ' );
33         } /* end if */
34     } /* end for */
35
36     putchar( '\n' );
37 } /* end function displayBits */
```

Bitwise Operators (Cont.)

- Example: fig10_07.c

```
18 void displayBits( unsigned value ) {
19     unsigned c; /* counter */
20
21     /* define displayMask and left shift 31 bits */
22     unsigned displayMask = 1 << 31;
23
24     printf( "%10u = ", value );
25
26     /* loop through bits */
27     for ( c = 1; c <= 32; c++ ) {
28         putchar( value & displayMask ? '1' : '0' );
29         value <<= 1; /* shift value left by 1 */
30
31         if ( c % 8 == 0 ) { /* output space after 8 bits */
32             putchar( ' ' );
33         } /* end if */
34     } /* end for */
35
36     putchar( '\n' );
37 } /* end function displayBits */
```

define a mask
10000000 00000000
00000000 00000000

Bitwise Operators (Cont.)

- Example: fig10_07.c

```
18 void displayBits( unsigned value ) {
19     unsigned c; /* counter */
20
21     /* define displayMask and left shift 31 bits */
22     unsigned displayMask = 1 << 31;
23
24     printf( "%10u = ", value );
25
26     /* loop through bits */
27     for ( c = 1; c <= 32; c++ ) {
28         putchar( value & displayMask ? '1' : '0' );
29         value <<= 1; /* shift value left by 1 */
30
31         if ( c % 8 == 0 ) { /* output space after 8 bits */
32             putchar( ' ' );
33         } /* end if */
34     } /* end for */
35
36     putchar( '\n' );
37 } /* end function displayBits */
```

define a mask
10000000 00000000
00000000 00000000

Bitwise Operators (Cont.)

- Example: fig10_07.c

```
18 void displayBits( unsigned value ) {
19    unsigned c; /* counter */
20
21    /* define displayMask and left shift 31 bits */
22    unsigned displayMask = 1 << 31;
23
24    printf( "%10u = ", value );
25
26    /* loop through bits */
27    for ( c = 1; c <= 32; c++ ) {
28       putchar( value & displayMask ? '1' : '0' );
29       value <<= 1; /* shift value left by 1 */
30
31       if ( c % 8 == 0 ) { /* output space after 8 bits */
32          putchar( ' ' );
33       } /* end if */
34    } /* end for */
35
36    putchar( '\n' );
37 } /* end function displayBits */
```

define a mask
10000000 00000000
00000000 00000000

all the bits except the high-order bit in **value** are "**masked off**", because any bit "ANDed" with 0 yields 0