Chapter 10 - C Structures, Unions, Bit Manipulations, and Enumerations

Outline

10.1 Introduction 10.2 **Structure Definitions** 10.3 **Initializing Structures** 10.4 **Accessing Members of Structures** 10.5 **Using Structures with Functions** typedef 10.6 **Example: High-Performance Card Shuffling and Dealing Simulation** 10.7 **Unions** 10.8 10.9 **Bitwise Operators** 10.10 Bit Fields 10.11 **Enumeration Constants**

[©] Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Objectives

- In this tutorial, you will learn:
 - To be able to create and use structures, unions and enumerations.
 - To be able to pass structures to functions call by value and call by reference.
 - To be able to manipulate data with the bitwise operators.
 - To be able to create bit fields for storing data compactly.

[©] Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



10.1 Introduction

Structures

- Collections of related variables (aggregates) under one name
 - Can contain variables of different data types
- Commonly used to define records to be stored in files
- Combined with pointers, can create linked lists, stacks, queues, and trees

[©] Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



Example

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

- 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



• struct information

- A struct cannot contain an instance of itself
- Can contain a member that is a pointer to the same structure type
- A structure definition does not reserve space in memory
 - Instead creates a new data type used to define structure variables

Definitions

– Defined like other variables:

```
card oneCard, deck[ 52 ], *cPtr;
```

Can use a comma separated list:

```
struct card {
    char *face;
    char *suit;
} oneCard, deck[ 52 ], *cPtr;
```

[©] Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



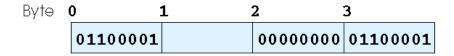


Fig. 10.1) A possible storage alignment for a variable of type struct example showing an undefined area in memory.§



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

[©] Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



10.3 Initializing Structures

- Initializer lists
 - Example:
 card oneCard = { "Three", "Hearts" };
- Assignment statements
 - Example:
 card threeHearts = oneCard;
 - Could also define and initialize threeHearts as follows:
 card threeHearts;

```
threeHearts.face = "Three";
threeHearts.suit = "Hearts";
```



10.4 Accessing Members of Structures

- Accessing structure members
 - Dot operator (.) used with structure variables
 card myCard;
 printf("%s", myCard.suit);
 Arrow operator (->) used with pointers to structure variables
 card *myCardPtr = &myCard;
 printf("%s", myCardPtr->suit);
 - myCardPtr->suit is equivalent to
 (*myCardPtr).suit



2)

fig10_02.c (Part 1 of

```
1 /* Fig. 10.2: fig10_02.c
      Using the structure member and
      structure pointer operators */
4 #include <stdio.h>
5
 /* card structure definition */
7 struct card {
      char *face; /* define pointer face */
8
      char *suit; /* define pointer suit */
10 }; /* end structure card */
11
12 int main()
13 {
      struct card a; /* define struct a */
14
15
      struct card *aPtr; /* define a pointer to card */
16
      /* place strings into card structures */
17
      a.face = "Ace";
18
      a.suit = "Spades";
19
20
      aPtr = &a; /* assign address of a to aPtr */
21
```

```
printf( "%s%s%s\n%s%s%s\n%s%s%s\n", a.face, " of ", a.suit,

aPtr->face, " of ", aPtr->suit,

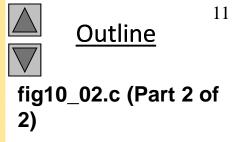
( *aPtr ).face, " of ", ( *aPtr ).suit );

return 0; /* indicates successful termination */

// 28

/* end main */

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



Program Output

10.5 Using Structures With Functions

- Passing structures to functions
 - Pass entire structure
 - Or, pass individual members
 - Both pass call by value
- To pass structures call-by-reference
 - Pass its address
 - Pass reference to it
- To pass arrays call-by-value
 - Create a structure with the array as a member
 - Pass the structure

10.6 typedef

typedef

- Creates synonyms (aliases) for previously defined data types
- Use typedef to create shorter type names
- Example:
 typedef struct Card *CardPtr;
- Defines a new type name CardPtr as a synonym for type struct Card *
- typedef does not create a new data type
 - Only creates an alias

10.7 Example: High-Performance Cardshuffling and Dealing Simulation

• Pseudocode:

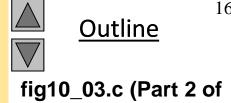
- Create an array of card structures
- Put cards in the deck
- Shuffle the deck
- Deal the cards



fig10_03.c (Part 1 of 4)

```
1 /* Fig. 10.3: fig10_03.c
      The card shuffling and dealing program using structures */
3 #include <stdio.h>
4 #include <stdlib.h>
  #include <time.h>
7 /* card structure definition */
 struct card {
      const char *face; /* define pointer face */
      const char *suit; /* define pointer suit */
10
11 }; /* end structure card */
12
13 typedef struct card Card;
14
15 /* prototypes */
16 void fillDeck( Card * const wDeck, const char * wFace[],
      const char * wSuit[] );
17
18 void shuffle( Card * const wDeck );
19 void deal( const Card * const wDeck );
20
21 int main()
22 {
      Card deck[ 52 ]; /* define array of Cards */
23
24
```

```
/* initialize array of pointers */
25
      const char *face[] = { "Ace", "Deuce", "Three", "Four", "Five",
26
         "Six" "Seven" "Eight" "Nine" "Ten"
27
         "Jack", "Queen", "King"};
28
29
      /* initialize array of pointers */
30
31
      const char *suit[] = { "Hearts", "Diamonds", "Clubs", "Spades"};
32
      srand( time( NULL ) ); /* randomize */
33
34
      fillDeck( deck, face, suit ); /* load the deck with Cards */
35
      shuffle( deck ); /* put Cards in random order */
36
      deal( deck ); /* deal all 52 Cards */
37
38
      return 0; /* indicates successful termination */
39
40
41 } /* end main */
42
43 /* place strings into Card structures */
44 void fillDeck( Card * const wDeck, const char * wFace[],
      const char * wSuit[] )
45
```



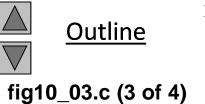
4)

46

int i; /* counter */

47

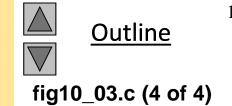
```
/* loop through wDeck */
49
      for (i = 0; i \le 51; i++) {
50
         wDeck[ i ].face = wFace[ i % 13 ];
51
         wDeck[ i ].suit = wSuit[ i / 13 ];
52
      } /* end for */
53
54
55 } /* end function fillDeck */
56
57 /* shuffle cards */
58 void shuffle( Card * const wDeck )
59 1
      int i; /* counter */
60
      int j; /* variable to hold random value between 0 - 51 */
61
      Card temp; /* define temporary structure for swapping Cards */
62
63
      /* loop through wDeck randomly swapping Cards */
64
      for (i = 0; i \le 51; i++) {
65
66
         j = rand() \% 52;
         temp = wDeck[ i ];
67
        wDeck[ i ] = wDeck[ j ];
68
         wDeck[ j ] = temp;
69
      } /* end for */
70
71
72 } /* end function shuffle */
```



```
74 /* deal cards */
75 void deal( const Card * const wDeck )
76 {
      int i; /* counter */
77
78
     /* loop through wDeck */
79
      for (i = 0; i \le 51; i++) {
80
         printf( "%5s of %-8s%c", wDeck[ i ].face, wDeck[ i ].suit,
81
            ( i + 1 ) % 2 ? '\t' : '\n' );
82
      } /* end for */
83
```

84

85 } /* end function deal */



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
		Spades			Diamonds
		Spades	Six	of	Hearts
Queen	of	Diamonds	Seven		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		Hearts
		Spades	Jack	of	Hearts
Deuce	of	Hearts	Jack		Clubs
Deuce	of	Spades	Ten	of	Diamonds
Seven	of	Spades	Nine		Clubs
		Clubs			Diamonds
Ten	of	Hearts	King	of	Diamonds



10.8 Unions

union

- Memory that contains a variety of objects over time
- Only contains one data member at a time
- Members of a union share space
- Conserves storage
- Only the last data member defined can be accessed

• union definitions

```
- Same as struct
    union Number {
        int x;
        float y;
    };
    union Number value;
```

[©] Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



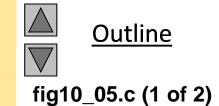
10.8 Unions

- Valid union operations
 - Assignment to union of same type: =
 - Taking address: &
 - Accessing union members: .
 - Accessing members using pointers: ->

[©] Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



```
1 /* Fig. 10.5: fig10_05.c
      An example of a union */
3 #include <stdio.h>
4
  /* number union definition */
 union number {
      int x; /* define int x */
      double y; /* define double y */
9 }; /* end union number */
10
11 int main()
12 {
      union number value; /* define union value */
13
14
      value.x = 100; /* put an integer into the union */
15
      printf( "%s\n%s\n%s%d\n%s%f\n\n",
16
             "Put a value in the integer member",
17
             "and print both members.",
18
             "int: ", value.x,
19
             "double:\n", value.y );
20
```



```
value.y = 100.0; /* put a double into the same union */
22
     printf( "%s\n%s\d\n%s%f\n",
23
          "Put a value in the floating member",
24
          "and print both members.".
25
          "int: ", value.x,
26
          "double:\n", value.y );
27
28
     return 0; /* indicates successful termination */
29
30
31 } /* end main */
Put a value in the integer member
and print both members.
int:
      100
double:
Put a value in the floating member
and print both members.
int:
      0
double:
100.000000
```

```
Outline
fig10_05.c (2 of 2)
```

10.9 Bitwise Operators

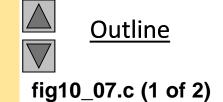
- All data represented internally as sequences of bits
 - Each bit can be either 0 or 1
 - Sequence of 8 bits forms a byte

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.
٨	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.
Fig. 10.6	The bitwise opera	ators.

[©] Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



```
1 /* Fig. 10.7: fig10_07.c
      Printing an unsigned integer in bits */
3 #include <stdio.h>
4
5 void displayBits( unsigned value ); /* prototype */
7 int main()
8 {
      unsigned x; /* variable to hold user input */
9
10
      printf( "Enter an unsigned integer: " );
11
      scanf( "%u", &x );
12
13
      displayBits( x );
14
15
      return 0; /* indicates successful termination */
16
17
18 } /* end main */
19
20 /* display bits of an unsigned integer value */
21 void displayBits( unsigned value )
22 {
```



23

24

unsigned c; /* counter */

```
/* define displayMask and left shift 31 bits */
25
      unsigned displayMask = 1 << 31;</pre>
26
27
      printf( "%7u = ", value );
28
29
      /* loop through bits */
30
      for (c = 1; c \le 32; c++) {
31
         putchar( value & displayMask ? '1' : '0' );
32
         value <<= 1; /* shift value left by 1 */</pre>
33
34
         if ( c \% 8 == 0 ) { /* output space after 8 bits */
35
            putchar( ' ');
36
         } /* end if */
37
38
      } /* end for */
39
40
      putchar( '\n' );
41
42 } /* end function displayBits */
Enter an unsigned integer: 65000
  65000 = 00000000 \ 00000000 \ 11111101 \ 11101000
```

```
Outline
fig10_07.c (2 of 2)
```

10.9 Bitwise Operators

Bit 1	Bit 2	Bit 1 & Bit 2
0	0	0
1	0	0
0	1	0
1	1	1
Fig. 10.8 Results of combining two bits with the bitwise AND operator &.		

[©] Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

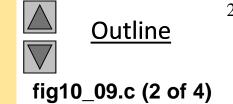


```
1 /* Fig. 10.9: fig10_09.c
      Using the bitwise AND, bitwise inclusive OR, bitwise
      exclusive OR and bitwise complement operators */
  #include <stdio.h>
5
  void displayBits( unsigned value ); /* prototype */
7
8 int main()
  {
9
      unsigned number1; /* define number1 */
10
      unsigned number2; /* define number2 */
11
      unsigned mask; /* define mask */
12
      unsigned setBits; /* define setBits */
13
14
      /* demonstrate bitwise & */
15
      number1 = 65535;
16
      mask = 1;
17
      printf( "The result of combining the following\n" );
18
      displayBits( number1 );
19
      displayBits( mask );
20
      printf( "using the bitwise AND operator & is\n" );
21
      displayBits( number1 & mask );
22
23
```



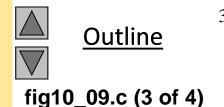
fig10_09.c (1 of 4)

```
/* demonstrate bitwise | */
24
      number1 = 15:
25
      setBits = 241;
26
27
      printf( "\nThe result of combining the following\n" );
      displayBits( number1 );
28
      displayBits( setBits );
29
      printf( "using the bitwise inclusive OR operator | is\n" );
30
      displayBits( number1 | setBits );
31
32
      /* demonstrate bitwise exclusive OR */
33
      number1 = 139;
34
      number2 = 199:
35
      printf( "\nThe result of combining the following\n" );
36
      displayBits( number1 );
37
      displayBits( number2 );
38
      printf( "using the bitwise exclusive OR operator ^ is\n" );
39
      displayBits( number1 ^ number2 );
40
41
      /* demonstrate bitwise complement */
42
      number1 = 21845;
43
      printf( "\nThe one's complement of\n" );
44
45
      displayBits( number1 );
      printf( "is\n" );
46
```



displayBits(~number1);

```
return 0; /* indicates successful termination */
49
50
51 } /* end main */
52
53 /* display bits of an unsigned integer value */
54 void displayBits(unsigned value)
55 {
56
      unsigned c; /* counter */
57
      /* declare displayMask and left shift 31 bits */
58
      unsigned displayMask = 1 << 31;</pre>
59
60
      printf( "%10u = ", value );
61
62
      /* loop through bits */
63
      for (c = 1; c \le 32; c++)
64
         putchar( value & displayMask ? '1' : '0' );
65
         value <<= 1; /* shift value left by 1 */</pre>
66
67
         if ( c % 8 == 0 ) { /* output a space after 8 bits */
68
            putchar( ' ');
69
         } /* end if */
70
71
```



} /* end for */

```
The result of combining the following
     65535 = 00000000 00000000 11111111 11111111
         1 = 00000000 00000000 00000000 00000001
using the bitwise AND operator & is
         1 = 00000000 00000000 00000000 00000001
The result of combining the following
        15 = 00000000 \ 00000000 \ 00000000 \ 00001111
       241 = 00000000 00000000 00000000 11110001
using the bitwise inclusive OR operator | is
       255 = 00000000 00000000 00000000 11111111
The result of combining the following
       139 = 00000000 00000000 00000000 10001011
       199 = 00000000 00000000 00000000 11000111
using the bitwise exclusive OR operator ^ is
        76 = 00000000 00000000 00000000 01001100
The one's complement of
     21845 = 00000000 00000000 01010101 01010101
is
```

putchar('\n');

75 } /* end function displayBits */

74

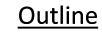


fig10_09.c (4 of 4) Program Output

4294945450 = 11111111 11111111 10101010 10101010

10.9 Bitwise Operators

Bit 1		Bit 2	Bit 1 Bit 2
0		0	0
1		0	1
0		1	1
1		1	1
Fig. 10.11	10.11 Posults of combining two bits with the bitwise inclusive OP		

Fig. 10.11 Results of combining two bits with the bitwise inclusive OR operator |.

[©] Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



10.9 Bitwise Operators

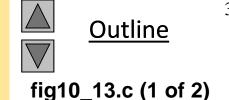
Bit 1	Bit 2	Bit 1 ^ Bit 2
0	0	0
1	0	1
0	1	1
1	1	0

Fig. 10.12 Results of combining two bits with the bitwise exclusive OR operator ^.

[©] Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



```
1 /* Fig. 10.13: fig10_13.c
      Using the bitwise shift operators */
 #include <stdio.h>
5 void displayBits( unsigned value ); /* prototype */
7 int main()
8
      unsigned number1 = 960; /* initialize number1 */
9
10
      /* demonstrate bitwise left shift */
11
      printf( "\nThe result of left shifting\n" );
12
      displayBits( number1 );
13
      printf( "8 bit positions using the " );
14
      printf( "left shift operator << is\n" );</pre>
15
      displayBits( number1 << 8 );</pre>
16
17
      /* demonstrate bitwise right shift */
18
      printf( "\nThe result of right shifting\n" );
19
      displayBits( number1 );
20
      printf( "8 bit positions using the " );
21
      printf( "right shift operator >> is\n" );
22
      displayBits( number1 >> 8 );
23
24
```



```
return 0; /* indicates successful termination */
26
27 } /* end main */
28
29 /* display bits of an unsigned integer value */
30 void displayBits( unsigned value )
31 {
      unsigned c; /* counter */
32
33
      /* declare displayMask and left shift 31 bits */
34
      unsigned displayMask = 1 << 31;</pre>
35
36
      printf( "%7u = ", value );
37
38
      /* loop through bits */
39
      for (c = 1; c \le 32; c++) {
40
         putchar( value & displayMask ? '1' : '0' );
41
         value <<= 1; /* shift value left by 1 */</pre>
42
43
         if (c \% 8 == 0) { /* output a space after 8 bits */
44
            putchar( ' ');
45
         } /* end if */
46
47
      } /* end for */
48
49
      putchar( '\n' );
50
51 } /* end function displayBits */
  © Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.
```



fig10_13.c (2 of 2)

```
The result of left shifting

960 = 00000000 00000000 00000011 11000000

8 bit positions using the left shift operator << is

245760 = 00000000 00000011 11000000 00000000
```



The result of right shifting 960 = 00000000 00000000 00000011 11000000 8 bit positions using the right shift operator >> is 3 = 00000000 00000000 000000011

10.9 Bitwise Operators

Bitwise assignment operators	
&=	Bitwise AND assignment operator.
=	Bitwise inclusive OR assignment operator.
Λ=	Bitwise exclusive OR assignment operator.
<<=	Left-shift assignment operator.
>>=	Right-shift assignment operator.
Fig. 10.14 The bitwise assignment	nt operators.

[©] Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



10.9 Bitwise Operators

Operator	Associativity	Туре
() []>	left to right	Highest
+ - ++! & * ~ sizeof (type)	right to left	Unary
* / %	left to right	multiplicative
+ -	left to right	additive
<< >>	left to right	shifting
< <= > >=	left to right	relational
== !=	left to right	equality
&	left to right	bitwise AND
Λ	left to right	bitwise OR
	left to right	bitwise OR
&&	left to right	logical AND
	left to right	logical OR
?:	right to left	conditional
= += -= *= /= &= = ^= <<= >>= %=	right to left	assignment
,	left to right	comma
Fig. 10.15 Operator precedence and associativity.		

[©] Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



10.10 Bit Fields

• Bit field

- Member of a structure whose size (in bits) has been specified
- Enable better memory utilization
- Must be defined as int or unsigned
- Cannot access individual bits

Defining bit fields

Follow unsigned or int member with a colon (:) and an integer constant representing the width of the field

```
– Example:
```

```
struct BitCard {
   unsigned face : 4;
   unsigned suit : 2;
   unsigned color : 1;
};
```

[©] Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

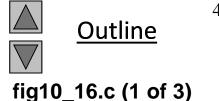


10.10 Bit Fields

- Unnamed bit field
 - Field used as padding in the structure
 - Nothing may be stored in the bits
 struct Example {
 unsigned a : 13;
 unsigned : 3;
 unsigned b : 4;
 }
 - Unnamed bit field with zero width aligns next bit field to a new storage unit boundary

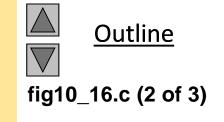


```
1 /* Fig. 10.16: fig10_16.c
      Representing cards with bit fields in a struct */
 #include <stdio.h>
  /* bitCard structure definition with bit fields */
7 struct bitCard {
      unsigned face : 4; /* 4 bits; 0-15 */
      unsigned suit : 2; /* 2 bits; 0-3 */
      unsigned color : 1; /* 1 bit; 0-1 */
10
11 }; /* end struct bitCard */
12
13 typedef struct bitCard Card;
14
15 void fillDeck( Card * const wDeck ); /* prototype */
16 void deal( const Card * const wDeck ); /* prototype */
17
18 int main()
19 {
      Card deck[ 52 ]; /* create array of Cards */
20
21
      fillDeck( deck );
22
23
      deal( deck );
24
```



return 0; /* indicates successful termination */

```
27 } /* end main */
28
29 /* initialize Cards */
30 void fillDeck( Card * const wDeck )
31 {
      int i; /* counter */
32
33
     /* loop through wDeck */
34
      for (i = 0; i \le 51; i++) {
35
         wDeck[ i ].face = i % 13;
36
         wDeck[ i ].suit = i / 13;
37
         wDeck[ i ].color = i / 26;
38
      } /* end for */
39
40
41 } /* end function fillDeck */
42
43 /* output cards in two column format; cards 0-25 subscripted with
      k1 (column 1); cards 26-51 subscripted k2 (column 2) */
44
45 void deal( const Card * const wDeck )
46 {
      int k1; /* subscripts 0-25 */
47
```



int k2; /* subscripts 26-51 */

```
/* loop through wDeck */
50
      for (k1 = 0, k2 = k1 + 26; k1 \le 25; k1++, k2++) {
51
         printf( "Card:%3d Suit:%2d Color:%2d ",
52
            wDeck[ k1 ].face, wDeck[ k1 ].suit, wDeck[ k1 ].color );
53
         printf( "Card:%3d Suit:%2d Color:%2d\n",
54
            wDeck[ k2 ].face, wDeck[ k2 ].suit, wDeck[ k2 ].color );
55
      } /* end for */
56
57
58 } /* end function deal */
```

```
Outline
fig10_16.c (3 of 3)
```

```
Suit: 0
                    Color: 0
                                                    Color: 1
Card:
       0
                               Card:
                                          Suit: 2
          Suit: 0
                    Color: 0
                                          Suit: 2
                                                    Color: 1
Card:
       1
                               Card:
                                       1
          Suit: 0
                    Color: 0
                                          Suit: 2
                                                    Color: 1
Card:
       2
                               Card:
                                       2
                    Color: 0
Card:
          Suit: 0
                                          Suit: 2
                                                    Color: 1
       3
                               Card:
Card:
          Suit: 0
                    Color: 0
                                          Suit: 2
                                                    Color: 1
       4
                               Card:
                    Color: 0
Card:
          Suit: 0
                               Card:
                                          Suit: 2
                                                    Color: 1
       5
          Suit: 0
                    Color: 0
                                          Suit: 2
                                                    Color: 1
Card:
                               Card:
          Suit: 0
                    Color: 0
                                                    Color: 1
Card:
       7
                               Card:
                                          Suit: 2
Card:
          Suit: 0
                    Color: 0
                                          Suit: 2
                                                    Color: 1
       8
                               Card:
                    Color: 0
Card:
      9
          Suit: 0
                               Card:
                                          Suit: 2
                                                    Color: 1
                                      9
Card: 10
          Suit: 0
                    Color: 0
                               Card: 10
                                          Suit: 2
                                                    Color: 1
Card: 11
          Suit: 0
                    Color: 0
                               Card: 11
                                          Suit: 2
                                                    Color: 1
                                                    Color: 1
Card: 12
          Suit: 0
                    Color: 0
                               Card: 12
                                          Suit: 2
Card:
          Suit: 1
                    Color: 0
                               Card:
                                          Suit: 3
                                                    Color: 1
       0
                    Color: 0
Card:
          Suit: 1
                               Card:
                                          Suit: 3
                                                    Color: 1
       1
                                       1
Card:
          Suit: 1
                    Color: 0
                                          Suit: 3
                                                    Color: 1
       2
                               Card:
Card:
       3
          Suit: 1
                    Color: 0
                               Card:
                                          Suit: 3
                                                    Color: 1
                                       3
Card:
       4
          Suit: 1
                    Color: 0
                               Card:
                                          Suit: 3
                                                    Color: 1
                    Color: 0
Card:
       5
          Suit: 1
                               Card:
                                          Suit: 3
                                                    Color: 1
Card:
          Suit: 1
                    Color: 0
                                          Suit: 3
                                                    Color: 1
       6
                               Card:
Card:
       7
          Suit: 1
                    Color: 0
                                          Suit: 3
                                                    Color: 1
                               Card:
Card:
       8
          Suit: 1
                    Color: 0
                               Card:
                                          Suit: 3
                                                    Color: 1
Card:
          Suit: 1
                    Color: 0
                               Card:
                                          Suit: 3
                                                    Color: 1
      9
Card: 10
          Suit: 1
                    Color: 0
                               Card: 10
                                          Suit: 3
                                                    Color: 1
Card: 11
          Suit: 1
                    Color: 0
                               Card: 11
                                          Suit: 3
                                                    Color: 1
Card: 12
          Suit: 1
                    Color: 0
                               Card: 12
                                                    Color: 1
                                          Suit: 3
```



10.11 Enumeration Constants

• Enumeration

- Set of integer constants represented by identifiers
- Enumeration constants are like symbolic constants whose values are automatically set
 - Values start at 0 and are incremented by 1
 - Values can be set explicitly with =
 - Need unique constant names
- Example:

```
enum Months { JAN = 1, 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 1 to 12
- Enumeration variables can only assume their enumeration constant values (not the integer representations)

[©] Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



```
1 /* Fig. 10.18: fig10_18.c
      Using an enumeration type */
3 #include <stdio.h>
4
  /* enumeration constants represent months of the year */
  enum months { JAN = 1, FEB, MAR, APR, MAY, JUN,
                 JUL, AUG, SEP, OCT, NOV, DEC };
7
8
9 int main()
10 {
      enum months month; /* can contain any of the 12 months */
11
12
      /* initialize array of pointers */
13
      const char *monthName[] = { "", "January", "February", "March",
14
         "April", "May", "June", "July", "August", "September", "October",
15
         "November", "December" };
16
17
      /* loop through months */
18
      for ( month = JAN; month <= DEC; month++ ) {</pre>
19
         printf( "%2d%11s\n", month, monthName[ month ] );
20
      } /* end for */
21
22
      return 0; /* indicates successful termination */
23
24 } /* end main */
```

```
Outline fig10_18.c
```

1	January
2	February
3	March
4	April
5	May
6	June
7	July
8	August
9	September
10	October
11	November
12	December

