Lekcija 10 - C Strukture, unije, manipulacija bitovima i enumeracije

Pregled

Ciljevi

- U ovoj lekciji
  - Naučićete da kreirate i koristite strukture, unije i enumeracije.
  - Predavaćete strukture kao argumente funkcija.
  - Koristićete bit-po-bit (bitwise) operatore.
  - Kreiraćete bit polja za kompaktno čuvanje podataka.

10.1 Uvod

- Strukture (Structures)
  - Kolekcije povezanih promjenljivih (agregata) pod jednim imenom
    - Mogu sadržati promjenljive različitih tipova
  - Uobičajeno se koriste za definisanje zapisa koji se čuvaju u datotekama
  - U kombinaciji sa pokazivačima, koriste se za kreiranje povezanih listi, stekova, redova i stabala

10.2 Definisanje strukture

- Primjer

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

  - `struct` uvodi definiciju strukture `card`
  - `card` je ime strukture i koristi se za deklarisanje promjenljivih tipa strukture
  - `card` sadrži 2 polja (člana) tipa `char *`
    - Data polja su `face` i `suit`

10.2 Definisanje strukture

- `struct` informacija
  - `struct` ne može sadržati referencu na samu sebe
  - Može sadržati polje koje je pokazivač na isti strukturni tip
  - Definicija strukture ne rezerviše memorijski prostor
    - Umjesto toga, kreira novi tip za definisanje strukturnih promjenljivih
- Definicije
  - Definisane kao ostale prromjenljive:

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

  - Može se koristiti lista razdvojena zarezima:

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

10.2 Definisanje strukture

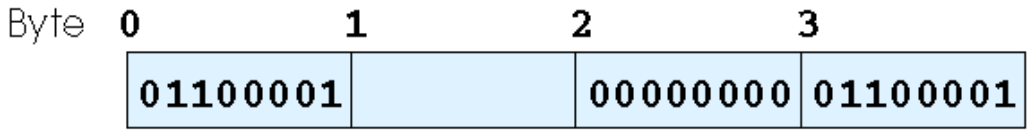


Fig. 10.1⟩ A possible storage alignment for a variable of type `struct example` showing an undefined area in memory.§

10.2 Definisanje strukture

- Validne operacije
  - Dodjeljivanje strukture strukturi istog tipa
  - Uzimanje adrese (&) strukture
  - Pristupanje poljima (članovima) strukture
  - Korišćenje `sizeof` operatora da bi se odredila veličina strukture (broj bajtova rezervisanih za strukturu)

10.3 Inicijalizacija strukture

- Liste inicijalizacije (Initializer lists)
  - Primjer:

```
card oneCard = { "Three", "Hearts" };
```

- Naredbe dodjeljivanja
  - Primjer:

```
card threeHearts = oneCard;
```

  - Moguće je definisati i inicijalizovati promjenljivu `threeHearts` na sledeći način:

```
card threeHearts;
threeHearts.face = “Three”;
threeHearts.suit = “Hearts”;
```

10.4 Pristup članovima strukture

- Pristup članovima strukture
  - Operator (`.`) zajedno sa strukturnim promjenljivim

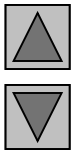
```
card myCard;
printf( "%s", myCard.suit );
```

  - Operator (**`->`**) se koristi za pokazivače na strukturu

```
card *myCardPtr = &myCard;
printf( "%s", myCardPtr->suit );
```

  - `myCardPtr->suit` je ekvivalentno sa

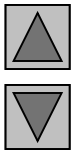
```
( *myCardPtr ).suit
```



Outline

fig10_02.c (Part 1 of 2)

```
1  /* Fig. 10.2: fig10_02.c
2     Using the structure member and
3     structure pointer operators */
4  #include <stdio.h>
5
6  /* card structure definition */
7  struct card {
8     char *face; /* define pointer face */
9     char *suit; /* define pointer suit */
10 }; /* end structure card */
11
12 int main()
13 {
14    struct card a;     /* define struct a */
15    struct card *aPtr; /* define a pointer to card */
16
17    /* place strings into card structures */
18    a.face = "Ace";
19    a.suit = "Spades";
20
21    aPtr = &a; /* assign address of a to aPtr */
22
```



Outline

```
23    printf( "%s%s%s\n%s%s%s\n%s%s%s\n", a.face, " of ", a.suit,
24          aPtr->face, " of ", aPtr->suit,
25          ( *aPtr ).face, " of ", ( *aPtr ).suit );
26
27    return 0; /* indicates successful termination */
28
29 } /* end main */
```

fig10_02.c (Part 2 of 2)

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

Program Output

10.5 Upotreba struktura sa funkcijama

- Strukture kao argumenti funkcija
  - Predavanje cijele strukture
    - ili, predavanje individualnih članova
  - U oba slučaja, predaju se po vrijednosti
- Strukture po referenci (call-by-reference)
  - Predaje se adresa strukture
  - Predaje se referenca na adresu
- Nizovi po vrijednosti (call-by-value)
  - Kreirajte strukturu koja ima niz kao član
  - Predajte strukturu

10.6 typedef

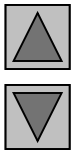
- typedef
  - Kreira sinonime (aliase) za već definisane tipove podataka
  - Koristite typedef za kreiranje kraćih imena tipova
  - Primjer:

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

  - Definiše novo ime tipe CardPtr kao sinonim za tip struct Card *
  - typedef ne kreira novi tip podataka
    - Samo se kreira alias

10.7 Primjer: podjela karata primjenom struktura

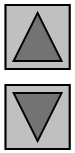
- Pseudo kod:
  - Kreirati niz struktura karte
  - Postaviti karte u špil
  - Promiješati špil
  - Podijelti karte



Outline

fig10_03.c (Part 1 of 4)

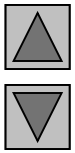
```
1  /* Fig. 10.3: fig10_03.c
2     The card shuffling and dealing program using structures */
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <time.h>
6
7  /* card structure definition */
8  struct card {
9     const char *face; /* define pointer face */
10    const char *suit; /* define pointer suit */
11 }; /* end structure card */
12
13 typedef struct card Card;
14
15 /* prototypes */
16 void fillDeck( Card * const wDeck, const char * wFace[],
17    const char * wSuit[] );
18 void shuffle( Card * const wDeck );
19 void deal( const Card * const wDeck );
20
21 int main()
22 {
23    Card deck[ 52 ]; /* define array of Cards */
24
```



Outline

fig10_03.c (Part 2 of 4)

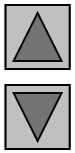
```
25    /* initialize array of pointers */
26    const char *face[] = { "Ace", "Deuce", "Three", "Four", "Five",
27       "Six", "Seven", "Eight", "Nine", "Ten",
28       "Jack", "Queen", "King"};
29
30    /* initialize array of pointers */
31    const char *suit[] = { "Hearts", "Diamonds", "Clubs", "Spades"};
32
33    srand( time( NULL ) ); /* randomize */
34
35    fillDeck( deck, face, suit ); /* load the deck with Cards */
36    shuffle( deck ); /* put Cards in random order */
37    deal( deck ); /* deal all 52 Cards */
38
39    return 0; /* indicates successful termination */
40
41 } /* end main */
42
43 /* place strings into Card structures */
44 void fillDeck( Card * const wDeck, const char * wFace[],
45    const char * wSuit[] )
46 {
47    int i; /* counter */
48
```



Outline

fig10_03.c (3 of 4)

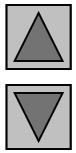
```
49    /* loop through wDeck */
50    for ( i = 0; i <= 51; i++ ) {
51       wDeck[ i ].face = wFace[ i % 13 ];
52       wDeck[ i ].suit = wSuit[ i / 13 ];
53    } /* end for */
54
55 } /* end function fillDeck */
56
57 /* shuffle cards */
58 void shuffle( Card * const wDeck )
59 {
60    int i;     /* counter */
61    int j;     /* variable to hold random value between 0 - 51 */
62    Card temp; /* define temporary structure for swapping Cards */
63
64    /* loop through wDeck randomly swapping Cards */
65    for ( i = 0; i <= 51; i++ ) {
66       j = rand() % 52;
67       temp = wDeck[ i ];
68       wDeck[ i ] = wDeck[ j ];
69       wDeck[ j ] = temp;
70    } /* end for */
71
72 } /* end function shuffle */
73
```



Outline

fig10_03.c (4 of 4)

```
74 /* deal cards */
75 void deal( const Card * const wDeck )
76 {
77    int i; /* counter */
78
79    /* loop through wDeck */
80    for ( i = 0; i <= 51; i++ ) {
81       printf( "%5s of %-8s%c", wDeck[ i ].face, wDeck[ i ].suit,
82          ( i + 1 ) % 2 ? '\t' : '\n' );
83    } /* end for */
84
85 } /* end function deal */
```



Outline

Program Output

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

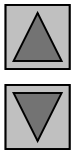
10.8 Unije (Unions)

- `union`
  - Memorija koja sadrži različite objekte u različitim vremenima
  - Sadrži samo jedan član u jednom trenutku
  - Članovi `union-a` dijele prostor
  - Čuva se memorijski prostor
  - Samo poslednji definisani član je dostupan
- Definicija `unije`
  - Isto kao struktura

```
union Number {
  int x;
  float y;
};
union Number value;
```

10.8 Unije

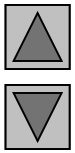
- Validne `union` operacije
  - Dodjeljivanje unije uniji istog tipa: `=`
  - Uzimanje adrese: `&`
  - Pristup članovima unije: `.`
  - Pristup članovima preko pokazivača: `->`



Outline

fig10_05.c (1 of 2)

```
1  /* Fig. 10.5: fig10_05.c
2     An example of a union */
3  #include <stdio.h>
4
5  /* number union definition */
6  union number {
7     int x;    /* define int x */
8     double y; /* define double y */
9  }; /* end union number */
10
11 int main()
12 {
13    union number value; /* define union value */
14
15    value.x = 100; /* put an integer into the union */
16    printf( "%s\n%s\n%s%d\n%s%f\n\n",
17            "Put a value in the integer member",
18            "and print both members.",
19            "int:   ", value.x,
20            "double:\n", value.y );
21
```



Outline

fig10_05.c (2 of 2)

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

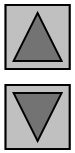
```
Put a value in the integer member
and print both members.
int:   100
double:
-92559592117433136000000000000000000000000000000000000000000000000.000000

Put a value in the floating member
and print both members.
int:   0
double:
100.000000
```

10.9 Bit-po-bit (bitwise) operatori

- Svi podaci su predstavljeni kao nizovi bitova
  - Svaki bit može biti 0 ili 1
  - Niz od 8 bitova formira jedan bajt

Operator		Opis
&	bitwise AND	Bitovi rezutata su 1 ako su odgovarajući bitovi u oba operanda 1.
|	bitwise inclusive OR	Bitovi rezutata su 1 ako je bar jedan od odgovarajućih bitova u operandima 1.
^	bitwise exclusive OR	Bitovi rezutata su 1 ako je tačno jedan od odgovarajućih bitova u operandima 1.
<<	left shift	Pomjera bitove prvog operanda ulijevo za broj bitova zadatih drugin operandom, popunjava nulama sa desne strane.
>>	right shift	Pomjera bitove prvog operanda udesno za broj bitova zadatih drugin operandom, popunjavanje sa lijeve strane je mašinski zavisno.
~	one's complement	Postavlja sve 0 na 1 i sve 1 na 0.

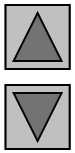
Fig. 10.6 Bit-po-bit(Bitwise) operatori.



Outline

fig10_07.c (1 of 2)

```
1  /* Fig. 10.7: fig10_07.c
2     Printing an unsigned integer in bits */
3  #include <stdio.h>
4
5  void displayBits( unsigned value ); /* prototype */
6
7  int main()
8  {
9     unsigned x; /* variable to hold user input */
10
11    printf( "Enter an unsigned integer: " );
12    scanf( "%u", &x );
13
14    displayBits( x );
15
16    return 0; /* indicates successful termination */
17
18 } /* end main */
19
20 /* display bits of an unsigned integer value */
21 void displayBits( unsigned value )
22 {
23    unsigned c; /* counter */
24
```



Outline

fig10_07.c (2 of 2)

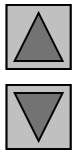
```
25    /* define displayMask and left shift 31 bits */
26    unsigned displayMask = 1 << 31;
27
28    printf( "%7u = ", value );
29
30    /* loop through bits */
31    for ( c = 1; c <= 32; c++ ) {
32       putchar( value & displayMask ? '1' : '0' );
33       value <<= 1; /* shift value left by 1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

```
Enter an unsigned integer: 65000
  65000 = 00000000 00000000 11111101 11101000
```

10.9 Bit-po-bit (bitwise) operatori

Bit 1	Bit 2	Bit 1 & Bit 2
0	0	0
1	0	0
0	1	0
1	1	1

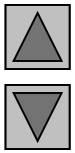
Fig. 10.8 Bitwise AND operator &.



Outline

fig10_09.c (1 of 4)

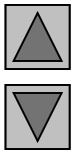
```
1  /* Fig. 10.9: fig10_09.c
2     Using the bitwise AND, bitwise inclusive OR, bitwise
3     exclusive OR and bitwise complement operators */
4  #include <stdio.h>
5
6  void displayBits( unsigned value ); /* prototype */
7
8  int main()
9  {
10    unsigned number1; /* define number1 */
11    unsigned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



Outline

fig10_09.c (2 of 4)

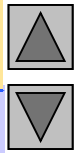
```
24    /* demonstrate bitwise | */
25    number1 = 15;
26    setBits = 241;
27    printf( "\nThe result of combining the following\n" );
28    displayBits( number1 );
29    displayBits( setBits );
30    printf( "using the bitwise inclusive OR operator | is\n" );
31    displayBits( number1 | setBits );
32
33    /* demonstrate bitwise exclusive OR */
34    number1 = 139;
35    number2 = 199;
36    printf( "\nThe result of combining the following\n" );
37    displayBits( number1 );
38    displayBits( number2 );
39    printf( "using the bitwise exclusive OR operator ^ is\n" );
40    displayBits( number1 ^ number2 );
41
42    /* demonstrate bitwise complement */
43    number1 = 21845;
44    printf( "\nThe one's complement of\n" );
45    displayBits( number1 );
46    printf( "is\n" );
47    displayBits( ~number1 );
48
```



Outline

fig10_09.c (3 of 4)

```
49    return 0; /* indicates successful termination */
50
51 } /* end main */
52
53 /* display bits of an unsigned integer value */
54 void displayBits( unsigned value )
55 {
56    unsigned c; /* counter */
57
58    /* declare displayMask and left shift 31 bits */
59    unsigned displayMask = 1 << 31;
60
61    printf( "%10u = ", value );
62
63    /* loop through bits */
64    for ( c = 1; c <= 32; c++ ) {
65       putchar( value & displayMask ? '1' : '0' );
66       value <<= 1; /* shift value left by 1 */
67
68       if ( c % 8 == 0 ) { /* output a space after 8 bits */
69          putchar( ' ' );
70       } /* end if */
71
72    } /* end for */
73
```



Outline

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

```
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
4294945450 = 11111111 11111111 10101010 10101010
```

fig10_09.c (4 of 4)

Program Output

10.9 Bit-po-bit (bitwise) operatori

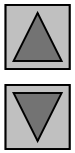
Bit 1	Bit 2	Bit 1 | Bit 2
0	0	0
1	0	1
0	1	1
1	1	1

Fig. 10.11 inclusive OR operator |.

10.9 Bit-po-bit (bitwise) operatori

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

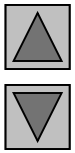
Fig. 10.12 Bitwise exclusive OR operator ^.



Outline

fig10_13.c (1 of 2)

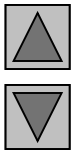
```
1  /* Fig. 10.13: fig10_13.c
2     Using the bitwise shift operators */
3  #include <stdio.h>
4
5  void displayBits( unsigned value ); /* prototype */
6
7  int main()
8  {
9     unsigned number1 = 960; /* initialize number1 */
10
11    /* demonstrate bitwise left shift */
12    printf( "\nThe result of left shifting\n" );
13    displayBits( number1 );
14    printf( "8 bit positions using the " );
15    printf( "left shift operator << is\n" );
16    displayBits( number1 << 8 );
17
18    /* demonstrate bitwise right shift */
19    printf( "\nThe result of right shifting\n" );
20    displayBits( number1 );
21    printf( "8 bit positions using the " );
22    printf( "right shift operator >> is\n" );
23    displayBits( number1 >> 8 );
24
```



Outline

fig10_13.c (2 of 2)

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



Outline

Program Output

```
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 00000000 00000011
```

10.9 Bit-po-bit (bitwise) operatori

Bitwise operatori dodjele	
&=	Bitwise AND assignment operator.
|=	Bitwise inclusive OR assignment operator.
^=	Bitwise exclusive OR assignment operator.
<<=	Left-shift assignment operator.
>>=	Right-shift assignment operator.

Fig. 10.14 Bitwise operatori dodjele (assignment operators).

10.9 Bit-po-bit (bitwise) operatori

Operator	Associativity	Type
`() [] . ->`	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
`^`	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.

10.10 Bit polja (bit fields)

- Bit polja
  - Članovi strukture čija je veličina (u bitovima) specificirana
  - Omogućavaju bolju upotrebu memorije
  - Moraju biti definisani kao `int` ili `unsigned`
  - Ne može se pristupati pojedinačnim bitovima
- Definisanje bit polja
  - Iza `unsigned` ili `int` ime člana sa dvotačkom (`:`) i cjelobrojnom konstantom koja predstavlja čirinu polja
  - Primjer:

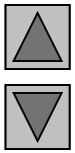
```
struct BitCard {
   unsigned face : 4;
   unsigned suit : 2;
   unsigned color : 1;
};
```

10.10 Bit polja

- Neimenovana bit polja
  - Polja koja se koriste kao dopuna (padding) u strukturi
  - Ništa se ne može smjestiti u njih

```
struct Example {
   unsigned a : 13;
   unsigned   : 3;
   unsigned b : 4;
}
```

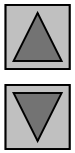
  - Neimenovana bit polja sa širinom polja 0 poravnavaju sledeće bit polje na novu granicu memorijske jedinice



Outline

fig10_16.c (1 of 3)

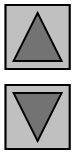
```
1  /* Fig. 10.16: fig10_16.c
2     Representing cards with bit fields in a struct */
3
4  #include <stdio.h>
5
6  /* bitCard structure definition with bit fields */
7  struct bitCard {
8     unsigned face : 4;  /* 4 bits; 0-15 */
9     unsigned suit : 2;  /* 2 bits; 0-3 */
10    unsigned color : 1; /* 1 bit; 0-1 */
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 {
20    Card deck[ 52 ]; /* create array of Cards */
21
22    fillDeck( deck );
23    deal( deck );
24
25    return 0; /* indicates successful termination */
26
```



Outline

fig10_16.c (2 of 3)

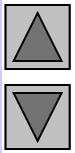
```
27 } /* end main */
28
29 /* initialize Cards */
30 void fillDeck( Card * const wDeck )
31 {
32    int i; /* counter */
33
34    /* loop through wDeck */
35    for ( i = 0; i <= 51; i++ ) {
36       wDeck[ i ].face = i % 13;
37       wDeck[ i ].suit = i / 13;
38       wDeck[ i ].color = i / 26;
39    } /* end for */
40
41 } /* end function fillDeck */
42
43 /* output cards in two column format; cards 0-25 subscripted with
44    k1 (column 1); cards 26-51 subscripted k2 (column 2) */
45 void deal( const Card * const wDeck )
46 {
47    int k1; /* subscripts 0-25 */
48    int k2; /* subscripts 26-51 */
49
```



Outline

fig10_16.c (3 of 3)

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



Outline

Program Output

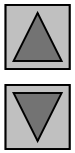
```
Card:  0  Suit: 0  Color: 0   Card:  0  Suit: 2  Color: 1
Card:  1  Suit: 0  Color: 0   Card:  1  Suit: 2  Color: 1
Card:  2  Suit: 0  Color: 0   Card:  2  Suit: 2  Color: 1
Card:  3  Suit: 0  Color: 0   Card:  3  Suit: 2  Color: 1
Card:  4  Suit: 0  Color: 0   Card:  4  Suit: 2  Color: 1
Card: 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

10.11 Enumeracije

- Enumeracije
  - Skup cjelobrojnih konstanti predstavljenih identifikatorima
  - Enumeracione konstante su kao simboličke konstante čije se vrijednosti automatski postavljaju
    - Vrijednosti počinju od 0 i inkrementiraju se za 1
    - Vrijednosti se mogu eksplicitno postaviti sa =
    - Potrebna su jedinstvena imena konstanti
  - Primjer:

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

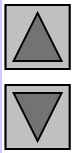
    - Kreira novi tip enum Months u kojem su identifikatori postavljeni na vrijednosti 1 - 12
  - Enumeracione promjenljive mogu dobiti samo vrijednost odgovarajuće konstante ali ne njene cjelobrojne reprezentacije



Outline

fig10_18.c

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



Outline

Program Output

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