Lekcija 7 – Pokazivači (pointers)

Pregled

7.1 Uvod
7.2 Deklaracija i inicijalizacija pokazivača
7.3 Operacije sa pokazivačima
7.4 Prenos argumenata po referenci
7.5 Upotreba const sa pokazivačima
7.6 Bubble Sort primjenom prenosa po referenci
7.7 Aritmetika pokazivača
7.8 Veza pokazivača i nizova
7.9 Nizovi pokazivača
7.10 Primjer: miješanje i podjela karata
7.11 Pokazivači na funkcije

Ciljevi lekcije

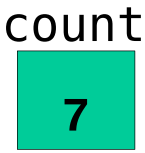
- U ovoj lekciji:
  - Naučićete upotrebu pokazivača.
  - Koristićete pokazivače za prenos argumenata po referenci.
  - Shvatićete tijesnu povezanost pokazivača, nizova i stringova.
  - Razumiješćete upotrebu pokazivača u funkcijama.
  - Znaćete da definišete i koristite nizove stringova.

7.1 Uvod

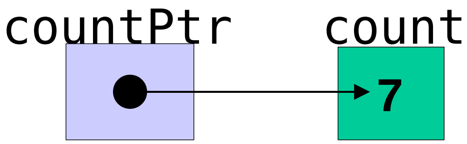
- Pokazivači (pointers)
  - Moćna mogućnost, ali složena
  - Simuliraju u C-u prenos po referenci (call-by-reference)
  - Tijesno su povezani sa nizovima i stringovima

7.2 Deklaracija i inicijalizacija pokazivača

- Pokazivačke promjenljive
  - Sadrže memorijsku adresu kao svoju vrijednost
  - Obične promjenljive sadrže određenu vrijednost (direct reference)



  - Pokazivači sadrže adresu promjenljive koja ima određenu vrijednost (indirect reference)
  - Indirekcija – referenciranje pokazivačke vrijednosti



7.2 Deklaracija i inicijalizacija pokazivača

- Definicija pokazivača (pointer definitions)
  - Koristimo simbol **`*`**

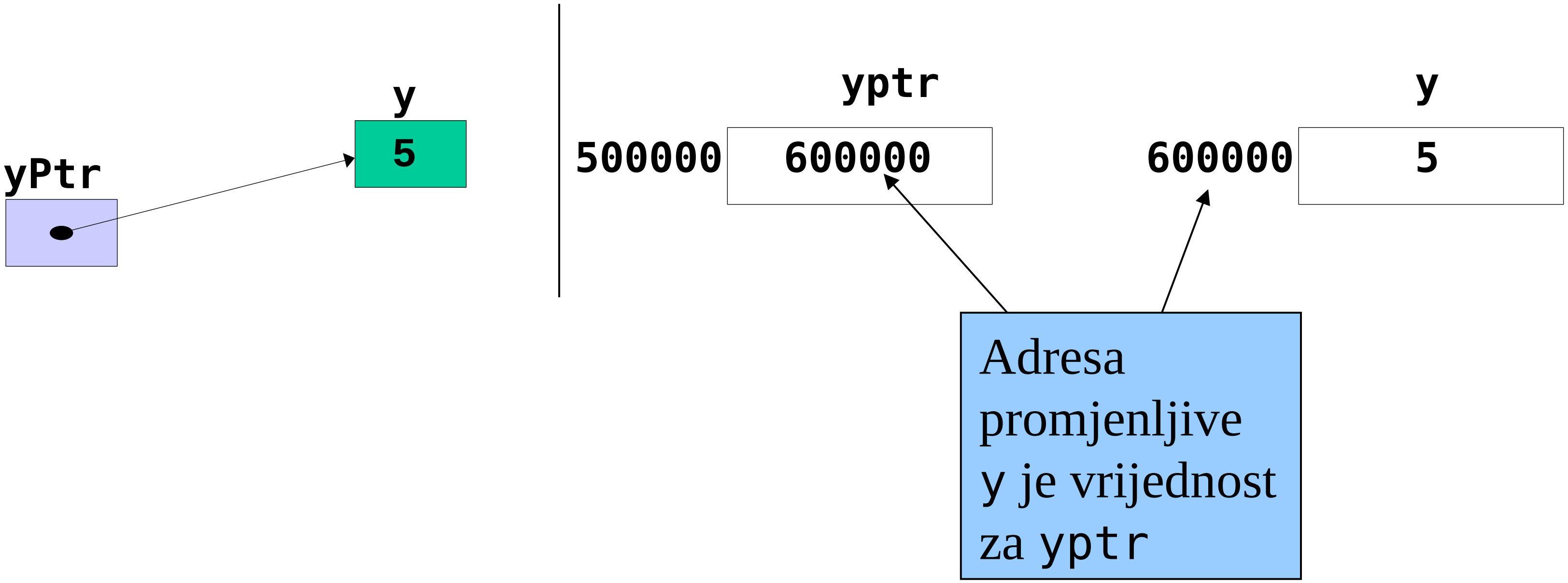
    ```
    int *myPtr;
    ```
  - Definiše pokazivač na `int` (pokazivač je tipa `int *`)
  - Više pokazivača u liniji - svaki mora da ima svoju *

    ```
    int *myPtr1, *myPtr2;
    ```
  - Možemo definisati pokazivač na bilo koji tip podatka
  - Pokazivač inicijalizujemo na `0`, `NULL` ili nekom adresom
    - `0` ili `NULL` – “points to nothing” (bolje je koristiti `NULL`)

7.3 Operacije sa pokazivačima

- & (operator adresiranja - address operator)
  - Vraća adresu operanda

```
int y = 5;
int *yPtr;
yPtr = &y;     /* yPtr dobija adresu y */
yPtr “pokazuje na” y
```



Adresa promjenljive y je vrijednost za `yptr`

7.3 Operacije sa pokazivačima

- * (operator indirekcije ili dereferenciranja - indirection/dereferencing operator)
  - Vraća sinonim/alias onog na šta pokazivač (operand) pokazuje
  - `*yptr` vraća `y` (jer `yptr` pokazuje na `y`)
  - * može biti korišćenja za dodjeljivanje
    - Vraća alias objekta

      ```
      *yptr = 7;  /* mijenja y na 7 */
      ```
  - Dereferencirani pokazivač (operand za *) mora biti lvalue (tj. ne može biti konstanta)
- * i & su međusobno inverzni

```c
/* Fig. 7.4: fig07_04.c
   Using the & and * operators */
#include <stdio.h>

int main()
{
   int a;         /* a is an integer */
   int *aPtr;     /* aPtr is a pointer to an integer */

   a = 7;
   aPtr = &a;     /* aPtr set to address of a */

   printf( "The address of a is %p"
           "\nThe value of aPtr is %p", &a, aPtr );

   printf( "\n\nThe value of a is %d"
           "\nThe value of *aPtr is %d", a, *aPtr );

   printf( "\n\nShowing that * and & are complements of "
           "each other\n&*aPtr = %p"
           "\n*&aPtr = %p\n", &*aPtr, *&aPtr );

   return 0; /* indicates successful termination */

} /* end main */
```

Adresa promjenljive `a` je vrijednost promjenjive `aPtr`.

`*` operator vraća alias onog na šta operand pokazuje. `aPtr` pokazuje na `a`, pa `*aPtr` vraća `a`.

Operatori `*` i `&` su međusobno inverzni

Program Output

```
The address of a is 0012FF7C
The value of aPtr is 0012FF7C

The value of a is 7
The value of *aPtr is 7

Showing that * and & are complements of each other.
&*aPtr = 0012FF7C
*&aPtr = 0012FF7C
```

7.3 Operacije sa pokazivačima

Operators								Associativity	Type
()	[]							left to right	highest
+	-	++	--	!	*	&	(type)	right to left	unary
*	/	%						left to right	multiplicative
+	-							left to right	additive
<	<=	>	>=					left to right	relational
==	!=							left to right	equality
&&								left to right	logical and
||								left to right	logical or
?:								right to left	conditional
=	+=	-=	*=	/=	%=			right to left	assignment
,								left to right	comma

Fig. 7.5 Operator precedence.

7.4 Prenos argumenata po referenci

- Prenos argumenata po referenci primjenom pokazivača
  - Predaje se adresa operanda (& operator)
  - Dozvoljava nam izmjenu vrijednosti u stvarnoj memorijskoj lokaciji
  - Nizovi se ne predaju sa & jer je ime niza već pokazivač
- * operator
  - Koristi se kao alias/nadimak za promjenljive unutar funkcije

```
void double( int *number )
 {
  *number = 2 * ( *number );
 }
```

  - `*number` je alias za predatu promjenljivu

fig07_06.c

```c
/* Fig. 7.6: fig07_06.c
   Cube a variable using call-by-value */
#include <stdio.h>

int cubeByValue( int n ); /* prototype */

int main()
{
   int number = 5; /* initialize number */

   printf( "The original value of number is %d", number );

   /* pass number by value to cubeByValue */
   number = cubeByValue( number );

   printf( "\nThe new value of number is %d\n", number );

   return 0; /* indicates successful termination */

} /* end main */

/* calculate and return cube of integer argument */
int cubeByValue( int n )
{
   return n * n * n;   /* cube local variable n and return result */

} /* end function cubeByValue */
```

```
The original value of number is 5
The new value of number is 125
```

Program Output

Outline

fig07_07.c

```
/* Fig. 7.7: fig07_07.c
   Cube a variable using call-by-reference with a pointer argument */

#include <stdio.h>

void cubeByReference( int *nPtr ); /* prototype */

int main()
{
   int number = 5; /* initialize number */

   printf( "The original value of number is %d", number );

   /* pass address of number to cubeByReference */
   cubeByReference( &number );

   printf( "\nThe new value of number is %d\n", number );

   return 0; /* indicates successful termination */

} /* end main */

/* calculate cube of *nPtr; modifies variable number in main */
void cubeByReference( int *nPtr )
{
   *nPtr = *nPtr * *nPtr * *nPtr;  /* cube *nPtr */
} /* end function cubeByReference */
```

Prototip funcije ima argument koji je pokazivač na int.

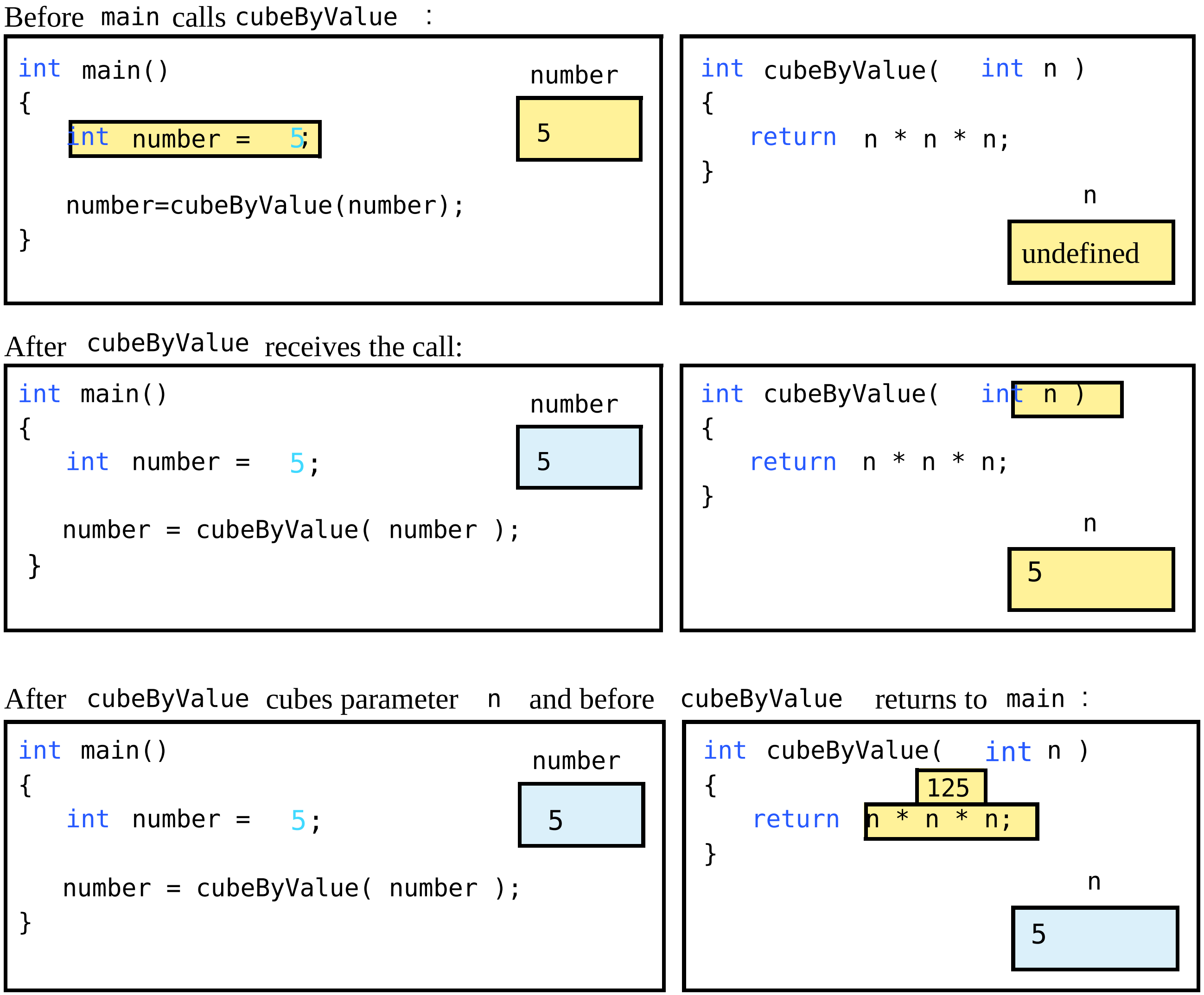
Stvarni argument je adresa datog broja - `cubeByReference` očekuje pokazivač.

Unutar `cubeByReference`, koristimo `*nPtr` (`*nPtr` je `number`).

```
The original value of number is 5
The new value of number is 125
```

Outline

Program Output



Before main calls cubeByValue :

```
int main()
{
   int number = 5;

   number=cubeByValue(number);
}
```

number

5

```
int cubeByValue( int n )
{
   return n * n * n;
}
```

n

undefined

After cubeByValue receives the call:

```
int main()
{
   int number = 5;

   number = cubeByValue( number );
}
```

number

5

```
int cubeByValue( int n )
{
   return n * n * n;
}
```

n

5

After cubeByValue cubes parameter n and before cubeByValue returns to main :

```
int main()
{
   int number = 5;

   number = cubeByValue( number );
}
```

number

5

```
int cubeByValue( int n )
{
   return n * n * n;
}
```

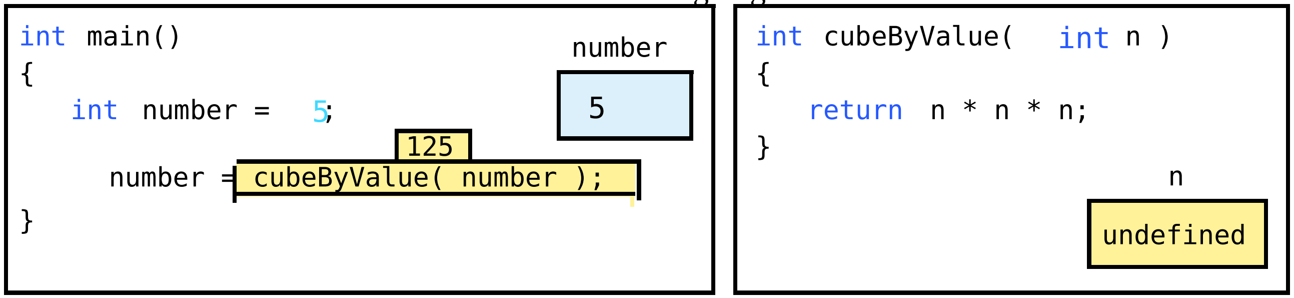
125

n

5

Fig. 7.8 Analiza prenosa argumenta po vrijednosti. (prvi dio)

After `cubeByValue` returns to `main` and before assigning the result to `number`:

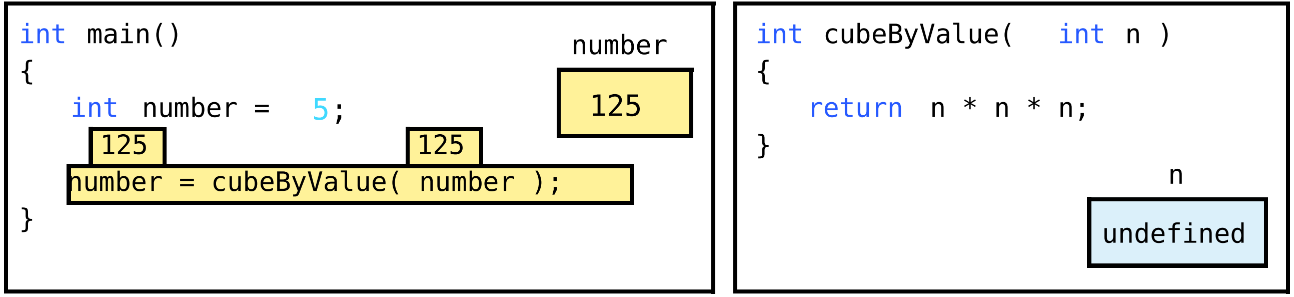


```
int main()
{
   int number = 5;

      number = cubeByValue( number );
}
```

```
int cubeByValue( int n )
{
   return n * n * n;
}
```

After `main` completes the assignment to `number`:

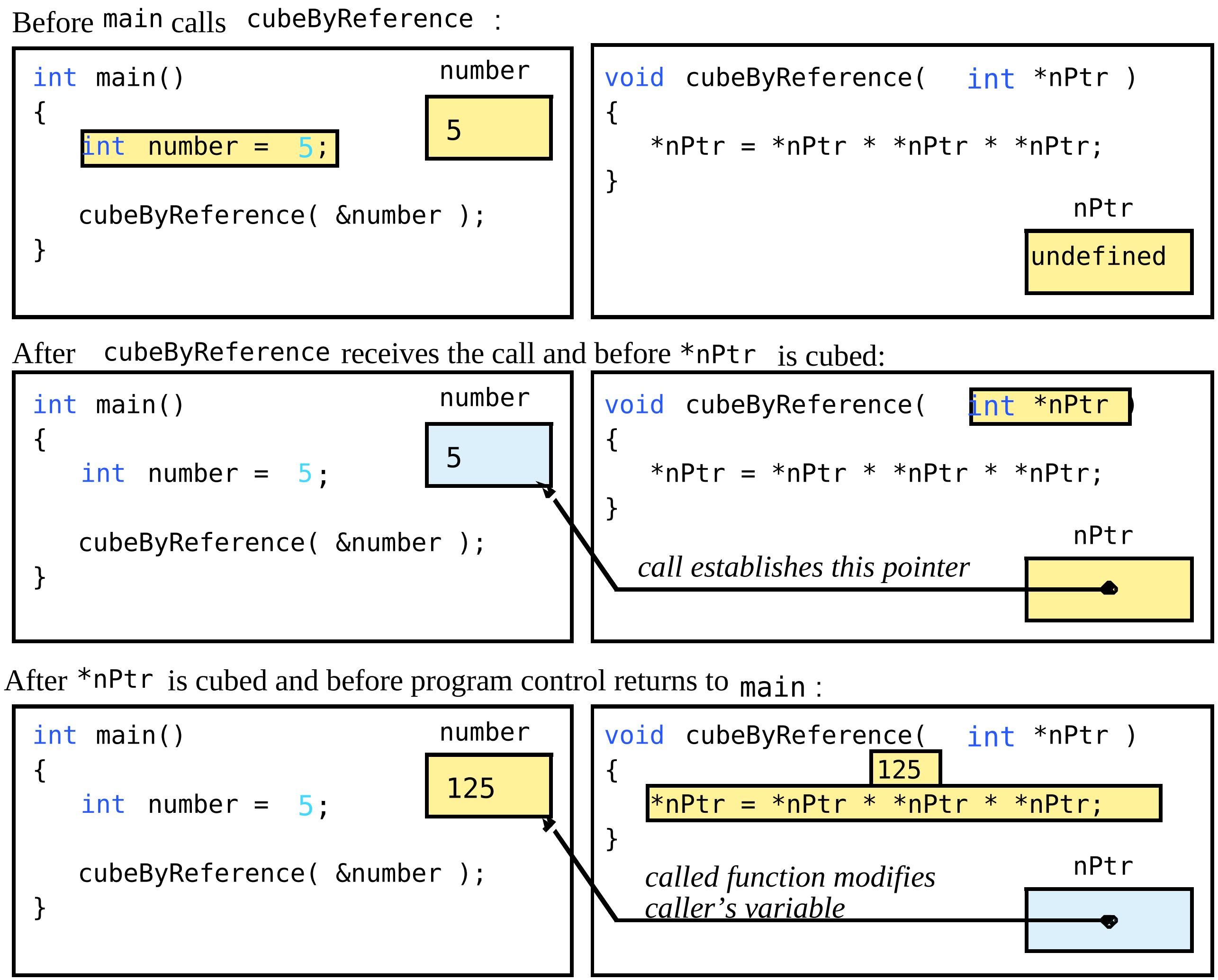


```
int main()
{
   int number = 5;

   number = cubeByValue( number );
}
```

```
int cubeByValue( int n )
{
   return n * n * n;
}
```

Fig. 7.8 Analiza prenosa argumenta po vrijednosti. (drugi dio)



Before `main` calls `cubeByReference` :

```
int main()
{
   int number = 5;

   cubeByReference( &number );
}
```

```
void cubeByReference( int *nPtr )
{
   *nPtr = *nPtr * *nPtr * *nPtr;
}
```

After `cubeByReference` receives the call and before `*nPtr` is cubed:

```
int main()
{
   int number = 5;

   cubeByReference( &number );
}
```

```
void cubeByReference( int *nPtr )
{
   *nPtr = *nPtr * *nPtr * *nPtr;
}
```

After `*nPtr` is cubed and before program control returns to `main` :

```
int main()
{
   int number = 5;

   cubeByReference( &number );
}
```

```
void cubeByReference( int *nPtr )
{
   *nPtr = *nPtr * *nPtr * *nPtr;
}
```

Fig. 7.9 Analiza prenosa argumenta po referenci. (argument je pokazivač).

7.5 Upotreba `const` sa pokazivačima

- `const` kvalifikator
  - Promjenljiva se ne može mijenjati
  - Koristite `const` ako funkcija ne treba da mijenja promjenljivu
  - Pokušaj promjene `const` promjenljive proizvodi grešku
- `const` pokazivači
  - Pokazuju na konstantnu memorijsku lokaciju
  - Moraju biti inicijalizovani pri definisanju
  - `int *const myPtr = &x;`
    - Tip `int *const` – konstantni pokazivač na `int`
  - `const int *myPtr = &x;`
    - Obični pokazivač na `const int`
  - `const int *const Ptr = &x;`
    - `const` pokazivač na `const int`
    - `x` možemo mijenjati ali ne i `*Ptr`

```c
/* Fig. 7.10: fig07_10.c
   Converting lowercase letters to uppercase letters
   using a non-constant pointer to non-constant data */

#include <stdio.h>
#include <ctype.h>

void convertToUppercase( char *sPtr ); /* prototype */

int main()
{
   char string[] = "characters and $32.98"; /* initialize char array */

   printf( "The string before conversion is: %s", string );
   convertToUppercase( string );
   printf( "\nThe string after conversion is: %s\n", string );

   return 0; /* indicates successful termination */

} /* end main */

```

Outline

fig07_10.c (Part 1 of 2)

```c
/* convert string to uppercase letters */
void convertToUppercase( char *sPtr )
{
   while ( *sPtr != '\0' ) { /* current character is not '\0' */

      if ( islower( *sPtr ) ) {    /* if character is lowercase, */
         *sPtr = toupper( *sPtr ); /* convert to uppercase */
      } /* end if */

      ++sPtr;  /* move sPtr to the next character */
   } /* end while */

} /* end function convertToUppercase */
```

```
The string before conversion is: characters and $32.98
The string after conversion is: CHARACTERS AND $32.98
```

Outline

fig07_10.c (Part 2 of 2)

Program Output

```c
/* Fig. 7.11: fig07_11.c
   Printing a string one character at a time using
   a non-constant pointer to constant data */

#include <stdio.h>

void printCharacters( const char *sPtr );

int main()
{
   /* initialize char array */
   char string[] = "print characters of a string";

   printf( "The string is:\n" );
   printCharacters( string );
   printf( "\n" );

   return 0; /* indicates successful termination */

} /* end main */

```

Outline

fig07_11.c (Part 1 of 2)

```c
/* sPtr cannot modify the character to which it points,
   i.e., sPtr is a "read-only" pointer */
void printCharacters( const char *sPtr )
{
   /* loop through entire string */
   for ( ; *sPtr != '\0'; sPtr++ ) { /* no initialization */
      printf( "%c", *sPtr );
   } /* end for */

} /* end function printCharacters */
```

```
The string is:
print characters of a string
```

fig07_11.c (Part 2 of 2)

Program Output

```c
/* Fig. 7.12: fig07_12.c
   Attempting to modify data through a
   non-constant pointer to constant data. */
#include <stdio.h>

void f( const int *xPtr ); /* prototype */

int main()
{
   int y;       /* define y */

   f( &y );     /* f attempts illegal modification */

   return 0;    /* indicates successful termination */

} /* end main */

/* xPtr cannot be used to modify the
   value of the variable to which it points */
void f( const int *xPtr )
{
   *xPtr = 100;  /* error: cannot modify a const object */
} /* end function f */
```

Outline

fig07_12.c

```
Compiling...
FIG07_12.c
d:\books\2003\chtp4\examples\ch07\fig07_12.c(22) : error C2166: l-value
   specifies const object
Error executing cl.exe.

FIG07_12.exe - 1 error(s), 0 warning(s)
```

Outline

Program Output

fig07_13.c

```c
/* Fig. 7.13: fig07_13.c
   Attempting to modify a constant pointer to non-constant data */
#include <stdio.h>

int main()
{
   int x; /* define x */
   int y; /* define y */

   /* ptr is a constant pointer to an integer that can be modified
      through ptr, but ptr always points to the same memory location */
   int * const ptr = &x;

   *ptr = 7; /* allowed: *ptr is not const */
   ptr = &y; /* error: ptr is const; cannot assign new address */

   return 0; /* indicates successful termination */

} /* end main */
```

Promjena `*ptr` je dozvoljena – `x` nije konstanta.

Promjena `ptr` je greška – `ptr` je konstantni pokazivač.

Program Output

```
Compiling...
FIG07_13.c
D:\books\2003\chtp4\Examples\ch07\FIG07_13.c(15) : error C2166: l-value
   specifies const object
Error executing cl.exe.

FIG07_13.exe - 1 error(s), 0 warning(s)
```

fig07_14.c

```
/* Fig. 7.14: fig07_14.c
   Attempting to modify a constant pointer to constant data. */
#include <stdio.h>

int main()
{
   int x = 5; /* initialize x */
   int y;     /* define y */

    /* ptr is a constant pointer to a constant integer. ptr always
       points to the same location; the integer at that location
       cannot be modified */
    const int *const ptr = &x;

    printf( "%d\n", *ptr );

    *ptr = 7; /* error: *ptr is const; cannot assign new value */
    ptr = &y; /* error: ptr is const; cannot assign new address */

    return 0; /* indicates successful termination */

 } /* end main */
```

Outline

Program Output

```
Compiling...
FIG07_14.c
D:\books\2003\chtp4\Examples\ch07\FIG07_14.c(17) : error C2166: l-value
   specifies const object
D:\books\2003\chtp4\Examples\ch07\FIG07_14.c(18) : error C2166: l-value
   specifies const object
Error executing cl.exe.

FIG07_12.exe - 2 error(s), 0 warning(s)
```

7.6 Bubble Sort primjenom prenosa argumenata po referenci

- Implementacija bubblesort-a primjenom pokazivača
  - Zamjena 2 elementa (swap)
  - `swap` funkcija mora da kao argument primi adresu (sa &) elementa niza
    - Elementi niza se predaju po vrijednosti
  - Primjenom pokazivača i operatora `*`, `swap` može zamijeniti mjesta bilo kojim elementima niza
- Psuedokod

```
Initialize array
    print data in original order
  Call function bubblesort
    print sorted array
  Define bubblesort
```

7.6 Bubble Sort primjenom prenosa argumenata po referenci

- `sizeof`
  - Vraća veličinu operanda u bajtima
  - Za nizove: veličina 1 elementa * broj elemenata
  - Ako je `sizeof( int )` `4` bajta, tada će

```
int myArray[ 10 ];
printf( "%d", sizeof( myArray ) );
```

    - Dati reazultat `40`
- `sizeof` možemo koristiti sa
  - Imenima promjenljivih
  - Imenima tipova (Type name)
  - Konstantama

```c
/* Fig. 7.15: fig07_15.c
   This program puts values into an array, sorts the values into
   ascending order, and prints the resulting array. */
#include <stdio.h>
#define SIZE 10

void bubbleSort( int *array, const int size ); /* prototype */

int main()
{
   /* initialize array a */
   int a[ SIZE ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };

   int i; /* counter */

   printf( "Data items in original order\n" );

   /* loop through array a */
   for ( i = 0; i < SIZE; i++ ) {
      printf( "%4d", a[ i ] );
   } /* end for */

   bubbleSort( a, SIZE ); /* sort the array */

   printf( "\nData items in ascending order\n" );

```

Outline

fig07_15.c (Part 1 of 3)

fig07_15.c (Part 2 of 3)

```c
   /* loop through array a */
   for ( i = 0; i < SIZE; i++ ) {
      printf( "%4d", a[ i ] );
   } /* end for */

   printf( "\n" );

   return 0; /* indicates successful termination */

} /* end main */

/* sort an array of integers using bubble sort algorithm */
void bubbleSort( int *array, const int size )
{
   void swap( int *element1Ptr, int *element2Ptr ); /* prototype */
   int pass; /* pass counter */
   int j;    /* comparison counter */

   /* loop to control passes */
   for ( pass = 0; pass < size - 1; pass++ ) {

      /* loop to control comparisons during each pass */
      for ( j = 0; j < size - 1; j++ ) {

         /* swap adjacent elements if they are out of order */
         if ( array[ j ] > array[ j + 1 ] ) {
            swap( &array[ j ], &array[ j + 1 ] );
         } /* end if */

      } /* end inner for */

   } /* end outer for */

} /* end function bubbleSort */

/* swap values at memory locations to which element1Ptr and
   element2Ptr point */
void swap( int *element1Ptr, int *element2Ptr )
{
   int hold = *element1Ptr;
   *element1Ptr = *element2Ptr;
   *element2Ptr = hold;
} /* end function swap */
```

```
Data items in original order
   2   6   4   8  10  12  89  68  45  37
Data items in ascending order
   2   4   6   8  10  12  37  45  68  89
```

Outline

fig07_15.c (Part 3 of 3)

Program Output

Outline

fig07_16.c

```c
/* Fig. 7.16: fig07_16.c
   Sizeof operator when used on an array name
   returns the number of bytes in the array. */
#include <stdio.h>

size_t getSize( float *ptr ); /* prototype */

int main()
{
   float array[ 20 ]; /* create array */

   printf( "The number of bytes in the array is %d"
           "\nThe number of bytes returned by getSize is %d\n",
           sizeof( array ), getSize( array ) );

   return 0; /* indicates successful termination */

} /* end main */

/* return size of ptr */
size_t getSize( float *ptr )
{
   return sizeof( ptr );

} /* end function getSize */
```

Program Output

```
The number of bytes in the array is 80
The number of bytes returned by getSize is 4
```

```c
/* Fig. 7.17: fig07_17.c
   Demonstrating the sizeof operator */
#include <stdio.h>

int main()
{
   char c;           /* define c */
   short s;          /* define s */
   int i;            /* define i */
   long l;           /* define l */
   float f;          /* define f */
   double d;         /* define d */
   long double ld;   /* define ld */
   int array[ 20 ];  /* initialize array */
   int *ptr = array; /* create pointer to array */

   printf( "     sizeof c = %d\tsizeof(char)  = %d"
           "\n     sizeof s = %d\tsizeof(short) = %d"
           "\n     sizeof i = %d\tsizeof(int) = %d"
           "\n     sizeof l = %d\tsizeof(long) = %d"
           "\n     sizeof f = %d\tsizeof(float) = %d"
           "\n     sizeof d = %d\tsizeof(double) = %d"
           "\n    sizeof ld = %d\tsizeof(long double) = %d"
           "\n sizeof array = %d"
           "\n   sizeof ptr = %d\n",
```

Outline

fig07_17.c (Part 1 of 2)

```c
          sizeof c, sizeof( char ), sizeof s,
          sizeof( short ), sizeof i, sizeof( int ),
          sizeof l, sizeof( long ), sizeof f,
          sizeof( float ), sizeof d, sizeof( double ),
          sizeof ld, sizeof( long double ),
          sizeof array, sizeof ptr );

   return 0; /* indicates successful termination */

} /* end main */
```

```
    sizeof c = 1       sizeof(char)  = 1
    sizeof s = 2       sizeof(short) = 2
    sizeof i = 4       sizeof(int) = 4
    sizeof l = 4       sizeof(long) = 4
    sizeof f = 4       sizeof(float) = 4
    sizeof d = 8       sizeof(double) = 8
   sizeof ld = 8       sizeof(long double) = 8
sizeof array = 80
  sizeof ptr = 4
```

fig07_17.c (Part 2 of 2)

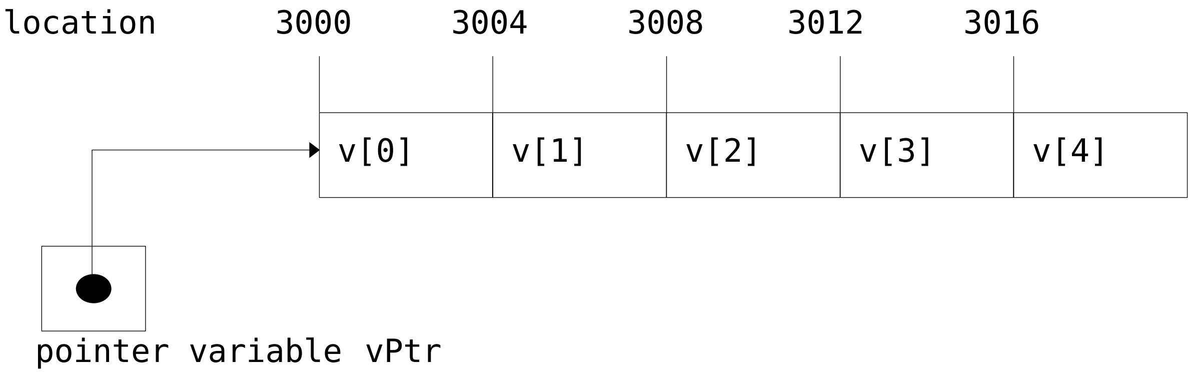
Program Output

7.7 Aritmetika pokazivača

- Možemo izvoditi neke aritmetičke operacije sa pokazivačima
  - Increment/decrement pokazivača (**++** ili **- -**)
  - Dodavanje broja pokazivaču (**+** ili **+=** , **-** ili **-=**)
  - Možemo oduzimati 2 pokazivača
  - Operacije najčešće nemaju smisla, osim ako se izvode nad nizovima

7.7 Aritmetika pokazivača

- 5-elementni `int` niz na mašini na kojoj je `sizeof(int)=4`
  - `vPtr` pokazuje na prvi element `v[ 0 ]`
    - Na lokaciji `3000 (vPtr = 3000)`
  - `vPtr += 2;` postavlja `vPtr` na `3008`
    - `vPtr` pokazuje na `v[ 2 ]` (inkrementiran za 2), ali je na ovoj mašini int 4 bajta, pa dobijamo adresu `3008`



7.7 Aritmetika pokazivača

- Oduzimanje pokazivača
  - Vraća broj elemenata. Ako je

    ```
    vPtr2 = v[ 2 ];
    vPtr = v[ 0 ];
    ```
  - `vPtr2 - vPtr` vraća rezultat 2
- Poređenje pokazivača (<, == , >)
  - Upoređuje koji pokazivač pokazuje na veći indeks u nizu
  - Takođe, provjera da li pokazivač pokazuje na NULL

7.7 Aritmetika pokazivača

- Pokazivači istog tipa mogu biti dodijeljeni jedan drugom
  - Ako nisu istog tipa, moramo izvršiti konverziju (cast)
  - Izuzetak od ovog pravila je pokazivač na `void` (tip `void *`)
    - Generički pokazivač, predstavlja bilo koji tip
    - Nije potrebna konverzija da bi preveli pokazivač u `void` pokazivač
    - `void` pokazivači se ne mogu dereferencirati

7.8 Veza između pokazivača i nizova

- Nizovi i pokazivači su tijesno povezani
  - Ime niza je kao konstantni pokazivač
  - Pokazivači mogu da zamijene indeksiranje niza
- Definišimo niz `b[ 5 ]` i pokazivač `bPtr`
  - Da pokazuju na isti objekta:

    ```
    bPtr = b;
    ```

    - Ime niza (`b`) je adresa adresa prvog elementa niza

    ```
    bPtr = &b[ 0 ]
    ```

    - Eksplicitno dodjeljivanje `bPtr` adresi prvog elementa niza `b`

7.8 Veza između pokazivača i nizova

- Element `b[ 3 ]`
  - Možemo mu pristupiti sa `*( bPtr + 3 )`
    - Ovdje je `n` offset. Ovo se naziva pointer/offset notacija
  - Možemo mu pristupiti sa `bptr[ 3 ]`
    - Ovo se naziva pointer/subscript notacija
    - `bPtr[ 3 ]` je isti što i `b[ 3 ]`
  - Možemo mu pristupiti sa `*( b + 3 )` – pokazivačka aritmetika nad nizom

```
/* Fig. 7.20: fig07_20.cpp
   Using subscripting and pointer notations with arrays */

#include <stdio.h>

int main()
{
   int b[] = { 10, 20, 30, 40 }; /* initialize array b */
   int *bPtr = b;                /* set bPtr to point to array b */
   int i;                        /* counter */
   int offset;                   /* counter */

   /* output array b using array subscript notation */
   printf( "Array b printed with:\nArray subscript notation\n" );

   /* loop through array b */
   for ( i = 0; i < 4; i++ ) {
      printf( "b[ %d ] = %d\n", i, b[ i ] );
   } /* end for */

   /* output array b using array name and pointer/offset notation */
   printf( "\nPointer/offset notation where\n"
           "the pointer is the array name\n" );

```

Outline

fig07_20.c (Part 1 of 2)

```
   /* loop through array b */
   for ( offset = 0; offset < 4; offset++ ) {
      printf( "*( b + %d ) = %d\n", offset, *( b + offset ) );
   } /* end for */

   /* output array b using bPtr and array subscript notation */
   printf( "\nPointer subscript notation\n" );

   /* loop through array b */
   for ( i = 0; i < 4; i++ ) {
      printf( "bPtr[ %d ] = %d\n", i, bPtr[ i ] );
   } /* end for */

   /* output array b using bPtr and pointer/offset notation */
   printf( "\nPointer/offset notation\n" );

   /* loop through array b */
   for ( offset = 0; offset < 4; offset++ ) {
      printf( "*( bPtr + %d ) = %d\n", offset, *( bPtr + offset ) );
   } /* end for */

   return 0; /* indicates successful termination */

} /* end main */
```

Outline

fig07_20.c (Part 2 of 2)

```
Array b printed with:
Array subscript notation
b[ 0 ] = 10
b[ 1 ] = 20
b[ 2 ] = 30
b[ 3 ] = 40

Pointer/offset notation where
the pointer is the array name
*( b + 0 ) = 10
*( b + 1 ) = 20
*( b + 2 ) = 30
*( b + 3 ) = 40

Pointer subscript notation
bPtr[ 0 ] = 10
bPtr[ 1 ] = 20
bPtr[ 2 ] = 30
bPtr[ 3 ] = 40

Pointer/offset notation
*( bPtr + 0 ) = 10
*( bPtr + 1 ) = 20
*( bPtr + 2 ) = 30
*( bPtr + 3 ) = 40
```

Program Output

fig07_21.c (Part 1 of 2)

```c
/* Fig. 7.21: fig07_21.c
   Copying a string using array notation and pointer notation. */
#include <stdio.h>

void copy1( char *s1, const char *s2 ); /* prototype */
void copy2( char *s1, const char *s2 ); /* prototype */

int main()
{
   char string1[ 10 ];          /* create array string1 */
   char *string2 = "Hello";     /* create a pointer to a string */
   char string3[ 10 ];          /* create array string3 */
   char string4[] = "Good Bye"; /* create a pointer to a string */

   copy1( string1, string2 );
   printf( "string1 = %s\n", string1 );

   copy2( string3, string4 );
   printf( "string3 = %s\n", string3 );

   return 0; /* indicates successful termination */

} /* end main */

```

```c
/* copy s2 to s1 using array notation */
void copy1( char *s1, const char *s2 )
{
   int i; /* counter */

   /* loop through strings */
   for ( i = 0; ( s1[ i ] = s2[ i ] ) != '\0'; i++ ) {
      ; /* do nothing in body */
   } /* end for */

} /* end function copy1 */

/* copy s2 to s1 using pointer notation */
void copy2( char *s1, const char *s2 )
{
   /* loop through strings */
   for ( ; ( *s1 = *s2 ) != '\0'; s1++, s2++ ) {
      ; /* do nothing in body */
   } /* end for */

} /* end function copy2 */
```

```
string1 = Hello
string3 = Good Bye
```

fig07_21.c (Part 2 of 2)

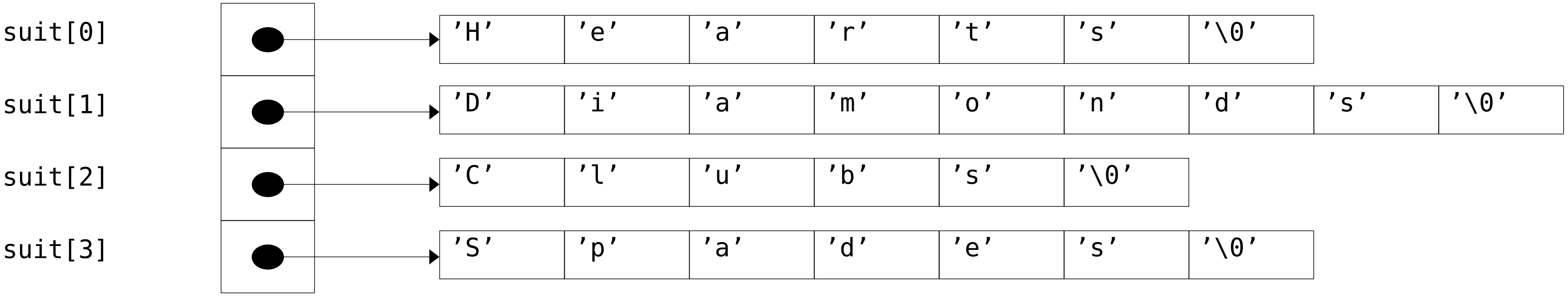
Program Output

7.9 Nizovi pokazivača

- Elementi niza mogu biti pokazivači
- Na primjer: niz stringova

```
char *suit[ 4 ] = { "Hearts", "Diamonds",
   "Clubs", "Spades" };
```

  - Stringovi su pokazivači na prvi karakter
  - `char *` – svaki element niza `suit` je pokazivač na `char`
  - Stringovi nisu stvarno smješteni u niz, već su smješteni samo pokazivači

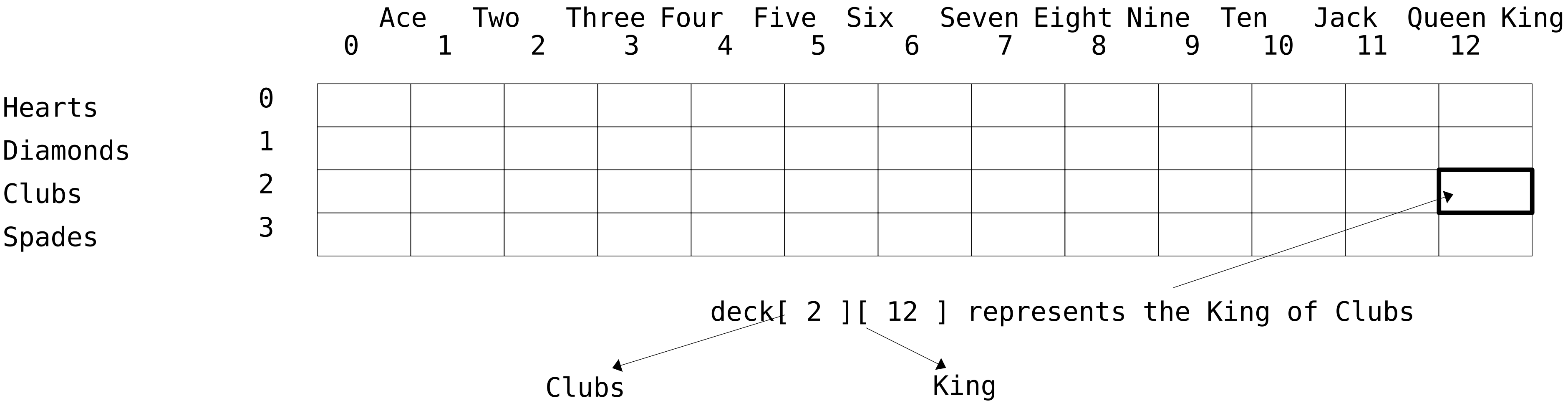


`suit[0]`	●	'H'	'e'	'a'	'r'	't'	's'	'\0'		
`suit[1]`	●	'D'	'i'	'a'	'm'	'o'	'n'	'd'	's'	'\0'
`suit[2]`	●	'C'	'l'	'u'	'b'	's'	'\0'			
`suit[3]`	●	'S'	'p'	'a'	'd'	'e'	's'	'\0'		

  - Niz `suit` ima fiksnu veličinu, ali stringovi mogu biti proizvoljne dužine

7.10 Primjer: miješanje i podjela karata

- Program za miješanje karata
  - Koristićemo niz pokazivača na stringove
  - Niz je dvodimenzionalni (suit, face)



deck[2][12] represents the King of Clubs

  - Brojeve 1-52 upisujemo u niz
    - Predstavljaju redosled u kom će karte biti podijeljene

7.10 Primjer: miješanje i podjela karata

- Pseudo kod
  - Opšti nivo (osnovni zadatak):

```
Promiješaj i podijeli 52 karte
```

  - Razlaganje na prvom nivou:

```
Inicijalizuj  niz suit
Inicijalizuj  niz face
Inicijalizuj  niz deck
Promiješaj deck
Podijeli 52 karte
```

7.10 Primjer: miješanje i podjela karata

- Razlaganje na drugom nivou
  - Razložiti *Promiješaj deck* na

```
Za svaku od 52 karte
    Postaviti broj karte u slučajno izabrani slobodni element (polje) niza deck
```

  - Razložiti *Podijeli 52 karte* na

```
Za svaku od 52 karte
    Pronaći broj karte u nizu deck i štampati vrstu i boju ( face i suit) karte
```

7.10 Primjer: miješanje i podjela karata

- Razlaganje na trećem nivou
  - Razložiti *Promiješaj deck* na

```
Izaberi polje deck-a slučajno
  While izabrano polje je ranije bilo odabrano
    Izaberi polje deck-a slučajno
  Postavi broj karte u izabrano polje
```

  - Razložiti *Podijeli 52 karte* na

```
Za svako polje niza deck
  If polje sadrži broj karte
    Štampati vrstu i boju (face i suit) karte
```

```c
/* Fig. 7.24: fig07_24.c
   Card shuffling dealing program */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/* prototypes */
void shuffle( int wDeck[][ 13 ] );
void deal( const int wDeck[][ 13 ], const char *wFace[],
           const char *wSuit[] );

int main()
{
   /* initialize suit array */
   const char *suit[ 4 ] = { "Hearts", "Diamonds", "Clubs", "Spades" };

   /* initialize face array */
   const char *face[ 13 ] =
      { "Ace", "Deuce", "Three", "Four",
        "Five", "Six", "Seven", "Eight",
        "Nine", "Ten", "Jack", "Queen", "King" };

   /* initialize deck array */
   int deck[ 4 ][ 13 ] = { 0 };

```

Outline

fig07_24.c (Part 1 of 4)

```
   srand( time( 0 ) ); /* seed random-number generator */

   shuffle( deck );
   deal( deck, face, suit );

   return 0; /* indicates successful termination */

} /* end main */

/* shuffle cards in deck */
void shuffle( int wDeck[][ 13 ] )
{
   int row;    /* row number */
   int column; /* column number */
   int card;   /* counter */

   /* for each of the 52 cards, choose slot of deck randomly */
   for ( card = 1; card <= 52; card++ ) {

      /* choose new random location until unoccupied slot found */
      do {
         row = rand() % 4;
         column = rand() % 13;
      } while( wDeck[ row ][ column ] != 0 ); /* end do...while */

```

Outline

fig07_24.c (Part 2 of 4)

```
      /* place card number in chosen slot of deck */
      wDeck[ row ][ column ] = card;
   } /* end for */

} /* end function shuffle */

/* deal cards in deck */
void deal( const int wDeck[][ 13 ], const char *wFace[],
           const char *wSuit[] )
{
   int card;   /* card counter */
   int row;    /* row counter */
   int column; /* column counter */

   /* deal each of the 52 cards */
   for ( card = 1; card <= 52; card++ ) {

      /* loop through rows of wDeck */
      for ( row = 0; row <= 3; row++ ) {

         /* loop through columns of wDeck for current row */
         for ( column = 0; column <= 12; column++ ) {

            /* if slot contains current card, display card */
            if ( wDeck[ row ][ column ] == card ) {
```

Outline

fig07_24.c (Part 3 of 4)

```c
               printf( "%5s of %-8s%c", wFace[ column ], wSuit[ row ],
                       card % 2 == 0 ? '\n' : '\t' );
            } /* end if */

         } /* end for */

      } /* end for */

   } /* end for */

} /* end function deal */
```

Outline

fig07_24.c (Part 4 of 4)

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

Program Output

7.11 Pokazivači na funkcije

- Pokazivač na funkciju
  - Sadrži adresu funkcije
  - Slično kao što je ime niza adresa prvog elementa, ime funkcije je početna adresa koda koji definiše funkciju
- Pokazivače na funkcije možemo
  - Predati kao argument funkcijama
  - Čuvati u elementima niza
  - Dodjeljivati drugim pokazivačima na funkcije

7.11 Pokazivači na funkcije

- Primjer: bubblesort
  - Funkija `bubble` ima argument koji je pokazivač na funkciju
    - `bubble` poziva tu pomoćnu funkciju
    - Ona određuje rastući ili opadajući poredak
  - Argument u `bubblesort` za ovaj pokazivač na funkciju

```
int ( *compare )( int a, int b )
```

govori `bubblesort-u` da očekuje pokazivač na funkciju koja ima 2 argumenta tipa `int` i vraća `int`

  - Ako nema lijeve zagrade, tj. ako je :

```
int *compare( int a, int b )
```

    - Definiše se funkcija koja ima 2 argumenta tipa `int` i vraća pokazivač na `int`

```
/* Fig. 7.26: fig07_26.c
   Multipurpose sorting program using function pointers */
#include <stdio.h>
#define SIZE 10

/* prototypes */
void bubble( int work[], const int size, int (*compare)( int a, int b ) );
int ascending( int a, int b );
int descending( int a, int b );

 int main()
 {
    int order;   /* 1 for ascending order or 2 for descending order */
    int counter; /* counter */

    /* initialize array a */
    int a[ SIZE ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };

    printf( "Enter 1 to sort in ascending order,\n"
            "Enter 2 to sort in descending order: " );
    scanf( "%d",  &order );

    printf( "\nData items in original order\n" );

```

fig07_26.c (Part 1 of 4)

fig07_26.c (Part 2 of 4)

```
   /* output original array */
   for ( counter = 0; counter < SIZE; counter++ ) {
      printf( "%5d", a[ counter ] );
   } /* end for */

   /* sort array in ascending order; pass function ascending as an
     argument to specify ascending sorting order */
   if ( order == 1 ) {
      bubble( a, SIZE, ascending );
      printf( "\nData items in ascending order\n" );
   } /* end if */
   else { /* pass function descending */
      bubble( a, SIZE, descending );
      printf( "\nData items in descending order\n" );
   } /* end else */

   /* output sorted array */
   for ( counter = 0; counter < SIZE; counter++ ) {
      printf( "%5d", a[ counter ] );
   } /* end for */

   printf( "\n" );

   return 0; /* indicates successful termination */

} /* end main */

```

```c
/* multipurpose bubble sort; parameter compare is a pointer to
   the comparison function that determines sorting order */
void bubble( int work[], const int size, int (*compare)( int a, int b ) )
{
   int pass;  /* pass counter */
   int count; /* comparison counter */

   void swap( int *element1Ptr, int *element2ptr ); /* prototype */

   /* loop to control passes */
   for ( pass = 1; pass < size; pass++ ) {

      /* loop to control number of comparisons per pass */
      for ( count = 0; count < size - 1; count++ ) {

         /* if adjacent elements are out of order, swap them */
         if ( (*compare)( work[ count ], work[ count + 1 ] ) ) {
            swap( &work[ count ], &work[ count + 1 ] );
         } /* end if */

      } /* end for */

   } /* end for */

} /* end function bubble */

```

fig07_26.c (Part 3 of 4)

```c
/* swap values at memory locations to which element1Ptr and
   element2Ptr point */
void swap( int *element1Ptr, int *element2Ptr )
{
   int hold; /* temporary holding variable */

   hold = *element1Ptr;
   *element1Ptr = *element2Ptr;
   *element2Ptr = hold;
} /* end function swap */

/* determine whether elements are out of order for an ascending
   order sort */
int ascending( int a, int b )
{
   return b < a; /* swap if b is less than a */

} /* end function ascending */

/* determine whether elements are out of order for a descending
   order sort */
int descending( int a, int b )
{
   return b > a; /* swap if b is greater than a */

} /* end function descending */
```

Outline

fig07_26.c (Part 4 of 4)

```
Enter 1 to sort in ascending order,
Enter 2 to sort in descending order: 1

Data items in original order
    2    6    4    8   10   12   89   68   45   37
Data items in ascending order
    2    4    6    8   10   12   37   45   68   89
```

```
Enter 1 to sort in ascending order,
Enter 2 to sort in descending order: 2

Data items in original order
    2    6    4    8   10   12   89   68   45   37
Data items in descending order
   89   68   45   37   12   10    8    6    4    2
```

Program Output