Computer Programming I

Ming-Feng Tsai (Victor Tsai)

Dept. of Computer Science
National Chengchi University

C Pointers

Using the **const** Qualifier with Pointers

- The **const** qualifier enables you to inform the compiler that the value of a particular variable should not be modified.

Using the **const** Qualifier with Pointers (Cont.)

- There are 4 ways to pass a pointer to a function:
  - **a non-constant pointer to non-constant data**
  - **a constant pointer to non-constant data**
  - **a non-constant pointer to constant data**
  - **a constant pointer to constant data**
- Each of the four combinations provides different access privileges

Using the **const** Qualifier with Pointers (Cont.)

- The **highest level of data access** is granted by a **non-constant pointer** to **non-constant data**.
- In this case, **the data can be modified** through the dereferenced pointer, and **the pointer can be modified to point to other data items.**
- A declaration for a non-constant pointer to non-constant data does not include **`const`**.
- Below we show a case of converting a string to uppercase by using a non-constant pointer to non-constant data

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_10.c

```
 6 #include <ctype.h>
 7
 8 void convertToUppercase( char *sPtr ); /* prototype */
 9
10 int main( void )
11 {
12    char string[] = "characters and $32.98"; /* initialize char array */
13
14    printf( "The string before conversion is: %s", string );
15    convertToUppercase( string );
16    printf( "\nThe string after conversion is: %s\n", string );
17    return 0; /* indicates successful termination */
18 } /* end main */
```

```
21 void convertToUppercase( char *sPtr )
22 {
23    while ( *sPtr != '\0' ) { /* current character is not '\0' */
24
25       if ( islower( *sPtr ) ) { /* if character is lowercase, */
26          *sPtr = toupper( *sPtr ); /* convert to uppercase */
27       } /* end if */
28
29       ++sPtr; /* move sPtr to the next character */
30    } /* end while */
31 } /* end function convertToUppercase */
```

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_10.c

```
 6 #include <ctype.h>
 7
 8 void convertToUppercase( char *sPtr ); /* prototype */
 9
10 int main( void )
11 {
12    char string[] = "characters and $32.98"; /* initialize char array */
13
14    printf( "The string before conversion is: %s", string );
15    convertToUppercase( string );
16    printf( "\nThe string after conversion is: %s\n", string );
17    return 0; /* indicates successful termination */
18 } /* end main */
```

```
21 void convertToUppercase( char *sPtr )
22 {
23    while ( *sPtr != '\0' ) { /* current character is not '\0' */
24
25       if ( islower( *sPtr ) ) { /* if character is lowercase, */
26          *sPtr = toupper( *sPtr ); /* convert to uppercase */
27       } /* end if */
28
29       ++sPtr; /* move sPtr to the next character */
30    } /* end while */
31 } /* end function convertToUppercase */
```

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_10.c

because of islower() and toupper()

```c
#include <ctype.h>

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

int main( void )
{
    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 */
```

```c
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 */
```

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_10.c

because of islower() and toupper()

```c
#include <ctype.h>

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

int main( void )
{
    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 */
```

```c
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 */
```

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_10.c

```c
#include <ctype.h>

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

int main( void )
{
    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 */
```

because of islower() and toupper()

declare a string

```c
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 */
```

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_10.c

```c
#include <ctype.h>

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

int main( void )
{
    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 */
```

because of islower() and toupper()

declare a string

```c
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 */
```

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_10.c

```
 6 #include <ctype.h>
 7
 8 void convertToUppercase( char *sPtr ); /* prototype */
 9
10 int main( void )
11 {
12    char string[] = "characters and $32.98"; /* initialize char array */
13
14    printf( "The string before conversion is: %s", string );
15    convertToUppercase( string );
16    printf( "\nThe string after conversion is: %s\n", string );
17    return 0; /* indicates successful termination */
18 } /* end main */
```

because of islower() and toupper()

declare a string

pass a non-constant pointer to the function

```
21 void convertToUppercase( char *sPtr )
22 {
23    while ( *sPtr != '\0' ) { /* current character is not '\0' */
24
25       if ( islower( *sPtr ) ) { /* if character is lowercase, */
26          *sPtr = toupper( *sPtr ); /* convert to uppercase */
27       } /* end if */
28
29       ++sPtr; /* move sPtr to the next character */
30    } /* end while */
31 } /* end function convertToUppercase */
```

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_10.c

```c
#include <ctype.h>

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

int main( void )
{
    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 */
```

because of islower() and toupper()

declare a string

pass a non-constant pointer to the function

```c
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 */
```

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_10.c

```
 6 #include <ctype.h>
 7
 8 void convertToUppercase( char *sPtr ); /* prototype */
 9
10 int main( void )
11 {
12    char string[] = "characters and $32.98"; /* initialize char array */
13
14    printf( "The string before conversion is: %s", string );
15    convertToUppercase( string );
16    printf( "\nThe string after conversion is: %s\n", string );
17    return 0; /* indicates successful termination */
18 } /* end main */
```

because of islower() and toupper()

declare a string

pass a non-constant pointer to the function

```
21 void convertToUppercase( char *sPtr )
22 {
23    while ( *sPtr != '\0' ) { /* current character is not '\0' */
24
25       if ( islower( *sPtr ) ) { /* if character is lowercase, */
26          *sPtr = toupper( *sPtr ); /* convert to uppercase */
27       } /* end if */
28
29       ++sPtr; /* move sPtr to the next character */
30    } /* end while */
31 } /* end function convertToUppercase */
```

receive a pointer

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_10.c

```c
#include <ctype.h>

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

int main( void )
{
    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 */
```

because of islower() and toupper()

declare a string

pass a non-constant pointer to the function

```c
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 */
```

receive a pointer

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_10.c

```c
#include <ctype.h>

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

int main( void )
{
    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 */
```

because of islower() and toupper()

declare a string

pass a non-constant pointer to the function

```c
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 */
```

receive a pointer

if character is lowercase, convert to uppercase

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_10.c

```
 6 #include <ctype.h>
 7
 8 void convertToUppercase( char *sPtr ); /* prototype */
 9
10 int main( void )
11 {
12    char string[] = "characters and $32.98"; /* initialize char array */
13
14    printf( "The string before conversion is: %s", string );
15    convertToUppercase( string );
16    printf( "\nThe string after conversion is: %s\n", string );
17    return 0; /* indicates successful termination */
18 } /* end main */
```

because of islower() and toupper()

declare a string

pass a non-constant pointer to the function

```
21 void convertToUppercase( char *sPtr )
22 {
23    while ( *sPtr != '\0' ) { /* current character is not '\0' */
24
25       if ( islower( *sPtr ) ) { /* if character is lowercase, */
26          *sPtr = toupper( *sPtr ); /* convert to uppercase */
27       } /* end if */
28
29       ++sPtr; /* move sPtr to the next character */
30    } /* end while */
31 } /* end function convertToUppercase */
```

receive a pointer

if character is lowercase, convert to uppercase

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_10.c

```
 6 #include <ctype.h>
 7
 8 void convertToUppercase( char *sPtr ); /* prototype */
 9
10 int main( void )
11 {
12    char string[] = "characters and $32.98"; /* initialize char array */
13
14    printf( "The string before conversion is: %s", string );
15    convertToUppercase( string );
16    printf( "\nThe string after conversion is: %s\n", string );
17    return 0; /* indicates successful termination */
18 } /* end main */
```

because of islower() and toupper()

declare a string

pass a non-constant pointer to the function

```
21 void convertToUppercase( char *sPtr )
22 {
23    while ( *sPtr != '\0' ) { /* current character is not '\0' */
24
25       if ( islower( *sPtr ) ) { /* if character is lowercase, */
26          *sPtr = toupper( *sPtr ); /* convert to uppercase */
27       } /* end if */
28
29       ++sPtr; /* move sPtr to the next character */
30    } /* end while */
31 } /* end function convertToUppercase */
```

receive a pointer

if character is lowercase, convert to uppercase

move the pointer to the next character

Using the **const** Qualifier with Pointers (Cont.)

- A **non-constant pointer** to **constant data** can be modified to point to any data item of the appropriate type, but the data to which it points cannot be modified.
- Such a pointer might be used to receive an array argument to a function that will process each element without modifying the data.

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_11.c

```
 7 void printCharacters( const char *sPtr );
 8
 9 int main( void )
10 {
11    /* initialize char array */
12    char string[] = "print characters of a string";
13
14    printf( "The string is:\n" );
15    printCharacters( string );
16    printf( "\n" );
17    return 0; /* indicates successful termination */
18 } /* end main */
19
20 /* sPtr cannot modify the character to which it points,
21    i.e., sPtr is a "read-only" pointer */
22 void printCharacters( const char *sPtr )
23 {
24    /* loop through entire string */
25    for ( ; *sPtr != '\0'; sPtr++ ) { /* no initialization */
26       printf( "%c", *sPtr );
27    } /* end for */
28 } /* end function printCharacters */
```

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_11.c

```
 7 void printCharacters( const char *sPtr );
 8
 9 int main( void )
10 {
11    /* initialize char array */
12    char string[] = "print characters of a string";
13
14    printf( "The string is:\n" );
15    printCharacters( string );
16    printf( "\n" );
17    return 0; /* indicates successful termination */
18 } /* end main */
19
20 /* sPtr cannot modify the character to which it points,
21    i.e., sPtr is a "read-only" pointer */
22 void printCharacters( const char *sPtr )
23 {
24    /* loop through entire string */
25    for ( ; *sPtr != '\0'; sPtr++ ) { /* no initialization */
26       printf( "%c", *sPtr );
27    } /* end for */
28 } /* end function printCharacters */
```

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_11.c

```
 7 void printCharacters( const char *sPtr );
 8
 9 int main( void )
10 {
11    /* initialize char array */
12    char string[] = "print characters of a string";
13
14    printf( "The string is:\n" );
15    printCharacters( string );
16    printf( "\n" );
17    return 0; /* indicates successful termination */
18 } /* end main */
19
20 /* sPtr cannot modify the character to which it points,
21    i.e., sPtr is a "read-only" pointer */
22 void printCharacters( const char *sPtr )
23 {
24    /* loop through entire string */
25    for ( ; *sPtr != '\0'; sPtr++ ) { /* no initialization */
26       printf( "%c", *sPtr );
27    } /* end for */
28 } /* end function printCharacters */
```

sPtr is a pointer to a character constant

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_11.c

```
 7 void printCharacters( const char *sPtr );
 8
 9 int main( void )
10 {
11    /* initialize char array */
12    char string[] = "print characters of a string";
13
14    printf( "The string is:\n" );
15    printCharacters( string );
16    printf( "\n" );
17    return 0; /* indicates successful termination */
18 } /* end main */
19
20 /* sPtr cannot modify the character to which it points,
21    i.e., sPtr is a "read-only" pointer */
22 void printCharacters( const char *sPtr )
23 {
24    /* loop through entire string */
25    for ( ; *sPtr != '\0'; sPtr++ ) { /* no initialization */
26       printf( "%c", *sPtr );
27    } /* end for */
28 } /* end function printCharacters */
```

sPtr is a pointer to a character constant

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_11.c

```
 7 void printCharacters( const char *sPtr );
 8
 9 int main( void )
10 {
11    /* initialize char array */
12    char string[] = "print characters of a string";
13
14    printf( "The string is:\n" );
15    printCharacters( string );
16    printf( "\n" );
17    return 0; /* indicates successful termination */
18 } /* end main */
19
20 /* sPtr cannot modify the character to which it points,
21    i.e., sPtr is a "read-only" pointer */
22 void printCharacters( const char *sPtr )
23 {
24    /* loop through entire string */
25    for ( ; *sPtr != '\0'; sPtr++ ) { /* no initialization */
26       printf( "%c", *sPtr );
27    } /* end for */
28 } /* end function printCharacters */
```

sPtr is a pointer to a character constant

the character pointed by sPtr cannot be modified

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_12.c

```
 6 void f( const int *xPtr ); /* prototype */
 7
 8 int main( void )
 9 {
10    int y; /* define y */
11
12    f( &y ); /* f attempts illegal modification */
13    return 0; /* indicates successful termination */
14 } /* end main */
15
16 /* xPtr cannot be used to modify the
17    value of the variable to which it points */
18 void f( const int *xPtr )
19 {
20    *xPtr = 100; /* error: cannot modify a const object */
21 } /* end function f */
```

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_12.c

```
 6 void f( const int *xPtr ); /* prototype */
 7
 8 int main( void )
 9 {
10    int y; /* define y */
11
12    f( &y ); /* f attempts illegal modification */
13    return 0; /* indicates successful termination */
14 } /* end main */
15
16 /* xPtr cannot be used to modify the
17    value of the variable to which it points */
18 void f( const int *xPtr )
19 {
20    *xPtr = 100; /* error: cannot modify a const object */
21 } /* end function f */
```

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_12.c

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

int main( void )
{
    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 */
```

xPtr is a pointer to an integer constant

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_12.c

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

int main( void )
{
    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 */
```

xPtr is a pointer to an integer constant

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_12.c

```
 6 void f( const int *xPtr ); /* prototype */
 7
 8 int main( void )
 9 {
10    int y; /* define y */
11
12    f( &y ); /* f attempts illegal modification */
13    return 0; /* indicates successful termination */
14 } /* end main */
15
16 /* xPtr cannot be used to modify the
17    value of the variable to which it points */
18 void f( const int *xPtr )
19 {
20    *xPtr = 100; /* error: cannot modify a const object */
21 } /* end function f */
```

xPtr is a pointer to an integer constant

invoke f( )

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_12.c

```
 6 void f( const int *xPtr ); /* prototype */
 7
 8 int main( void )
 9 {
10    int y; /* define y */
11
12    f( &y ); /* f attempts illegal modification */
13    return 0; /* indicates successful termination */
14 } /* end main */
15
16 /* xPtr cannot be used to modify the
17    value of the variable to which it points */
18 void f( const int *xPtr )
19 {
20    *xPtr = 100; /* error: cannot modify a const object */
21 } /* end function f */
```

xPtr is a pointer to an integer constant

invoke f( )

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_12.c

```
 6 void f( const int *xPtr ); /* prototype */
 7
 8 int main( void )
 9 {
10    int y; /* define y */
11
12    f( &y ); /* f attempts illegal modification */
13    return 0; /* indicates successful termination */
14 } /* end main */
15
16 /* xPtr cannot be used to modify the
17    value of the variable to which it points */
18 void f( const int *xPtr )
19 {
20    *xPtr = 100; /* error: cannot modify a const object */
21 } /* end function f */
```

xPtr is a pointer to an integer constant

invoke f( )

error: cannot modify a const object!!

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_12.c

```
 6 void f( const int *xPtr ); /* prototype */
 7
 8 int main( void )
 9 {
10    int y; /* define y */
11
12    f( &y ); /* f attempts illegal modification */
13    return 0; /* indicates successful termination */
14 } /* end main */
15
16 /* xPtr cannot be used to modify the
17    value of the variable to which it points */
18 void f( const int *xPtr )
19 {
20    *xPtr = 100; /* error: cannot modify a const object */
21 } /* end function f */
```

xPtr is a pointer to an integer constant

invoke f( )

error: cannot modify a const object!!

```
fig07_12.c: In function ‘f’:
fig07_12.c:20: error: assignment of read-only location
```

Using the **const** Qualifier with Pointers (Cont.)

- A **constant pointer to non-constant data** always points to the same memory location, and the data at that location can be modified through the pointer.
  - This is the default for an array name.
    - An array name is a constant pointer to the beginning of the array.
    - All data in the array can be accessed and changed by using the array name and array subscripting.

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_13.c

```
5  int main( void )
6  {
7      int x; /* define x */
8      int y; /* define y */
9
10     /* ptr is a constant pointer to an integer that can be modified
11        through ptr, but ptr always points to the same memory location */
12     int * const ptr = &x;
13
14     *ptr = 7; /* allowed: *ptr is not const */
15     ptr = &y; /* error: ptr is const; cannot assign new address */
16     return 0; /* indicates successful termination */
17 } /* end main */
```

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_13.c

```
5  int main( void )
6  {
7      int x; /* define x */
8      int y; /* define y */
9
10     /* ptr is a constant pointer to an integer that can be modified
11        through ptr, but ptr always points to the same memory location */
12     int * const ptr = &x;
13
14     *ptr = 7; /* allowed: *ptr is not const */
15     ptr = &y; /* error: ptr is const; cannot assign new address */
16     return 0; /* indicates successful termination */
17 } /* end main */
```

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_13.c

```
5  int main( void )
6  {
7      int x; /* define x */
8      int y; /* define y */
9
10     /* ptr is a constant pointer to an integer that can be modified
11        through ptr, but ptr always points to the same memory location */
12     int * const ptr = &x;
13
14     *ptr = 7; /* allowed: *ptr is not const */
15     ptr = &y; /* error: ptr is const; cannot assign new address */
16     return 0; /* indicates successful termination */
17 } /* end main */
```

ptr is a constant pointer to an integer; ptr is initialized with the address of x

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_13.c

```
 5 int main( void )
 6 {
 7    int x; /* define x */
 8    int y; /* define y */
 9
10    /* ptr is a constant pointer to an integer that can be modified
11       through ptr, but ptr always points to the same memory location */
12    int * const ptr = &x;
13
14    *ptr = 7; /* allowed: *ptr is not const */
15    ptr = &y; /* error: ptr is const; cannot assign new address */
16    return 0; /* indicates successful termination */
17 } /* end main */
```

ptr is a constant pointer to an integer; ptr is initialized with the address of x

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_13.c

```
5  int main( void )
6  {
7      int x; /* define x */
8      int y; /* define y */
9
10     /* ptr is a constant pointer to an integer that can be modified
11        through ptr, but ptr always points to the same memory location */
12     int * const ptr = &x;
13
14     *ptr = 7; /* allowed: *ptr is not const */
15     ptr = &y; /* error: ptr is const; cannot assign new address */
16     return 0; /* indicates successful termination */
17 } /* end main */
```

ptr is a constant pointer to an integer; ptr is initialized with the address of x

error: ptr is const; cannot assign new address!!

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_13.c

```
 5 int main( void )
 6 {
 7    int x; /* define x */
 8    int y; /* define y */
 9
10    /* ptr is a constant pointer to an integer that can be modified
11       through ptr, but ptr always points to the same memory location */
12    int * const ptr = &x;
13
14    *ptr = 7; /* allowed: *ptr is not const */
15    ptr = &y; /* error: ptr is const; cannot assign new address */
16    return 0; /* indicates successful termination */
17 } /* end main */
```

ptr is a constant pointer to an integer; ptr is initialized with the address of x

error: ptr is const; cannot assign new address!!

```
fig07_13.c: In function ‘main’:
fig07_13.c:15: error: assignment of read-only variable ‘ptr’
```

Using the **const** Qualifier with Pointers (Cont.)

- The least access privilege is granted by a **constant pointer to constant data.**
- Such a pointer always points to the same memory location, and the data at that memory location cannot be modified.
- This is how an array should be passed to a function that only looks at the array using array subscript notation and does not modify the array.

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_14.c

```
 5 int main( void )
 6 {
 7    int x = 5; /* initialize x */
 8    int y; /* define y */
 9
10    /* ptr is a constant pointer to a constant integer. ptr always
11       points to the same location; the integer at that location
12       cannot be modified */
13    const int *const ptr = &x;
14
15    printf( "%d\n", *ptr );
16
17    *ptr = 7; /* error: *ptr is const; cannot assign new value */
18    ptr = &y; /* error: ptr is const; cannot assign new address */
19    return 0; /* indicates successful termination */
20 } /* end main */
```

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_14.c

```
 5 int main( void )
 6 {
 7    int x = 5; /* initialize x */
 8    int y; /* define y */
 9
10    /* ptr is a constant pointer to a constant integer. ptr always
11       points to the same location; the integer at that location
12       cannot be modified */
13    const int *const ptr = &x;
14
15    printf( "%d\n", *ptr );
16
17    *ptr = 7; /* error: *ptr is const; cannot assign new value */
18    ptr = &y; /* error: ptr is const; cannot assign new address */
19    return 0; /* indicates successful termination */
20 } /* end main */
```

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_14.c

```
 5 int main( void )
 6 {
 7    int x = 5; /* initialize x */
 8    int y; /* define y */
 9
10    /* ptr is a constant pointer to a constant integer. ptr always
11       points to the same location; the integer at that location
12       cannot be modified */
13    const int *const ptr = &x;
14
15    printf( "%d\n", *ptr );
16
17    *ptr = 7; /* error: *ptr is const; cannot assign new value */
18    ptr = &y; /* error: ptr is const; cannot assign new address */
19    return 0; /* indicates successful termination */
20 } /* end main */
```

ptr is a constant pointer to an integer constant

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_14.c

```
 5 int main( void )
 6 {
 7    int x = 5; /* initialize x */
 8    int y; /* define y */
 9
10    /* ptr is a constant pointer to a constant integer. ptr always
11       points to the same location; the integer at that location
12       cannot be modified */
13    const int *const ptr = &x;
14
15    printf( "%d\n", *ptr );
16
17    *ptr = 7; /* error: *ptr is const; cannot assign new value */
18    ptr = &y; /* error: ptr is const; cannot assign new address */
19    return 0; /* indicates successful termination */
20 } /* end main */
```

ptr is a constant pointer to an integer constant

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_14.c

```
 5 int main( void )
 6 {
 7    int x = 5; /* initialize x */
 8    int y; /* define y */
 9
10    /* ptr is a constant pointer to a constant integer. ptr always
11       points to the same location; the integer at that location
12       cannot be modified */
13    const int *const ptr = &x;
14
15    printf( "%d\n", *ptr );
16
17    *ptr = 7; /* error: *ptr is const; cannot assign new value */
18    ptr = &y; /* error: ptr is const; cannot assign new address */
19    return 0; /* indicates successful termination */
20 } /* end main */
```

ptr is a constant pointer to an integer constant

error: *ptr is const; cannot assign new value
error: ptr is const; cannot assign new address

Using the **const** Qualifier with Pointers (Cont.)

- Example: fig07_14.c

```
 5 int main( void )
 6 {
 7    int x = 5; /* initialize x */
 8    int y; /* define y */
 9
10    /* ptr is a constant pointer to a constant integer. ptr always
11       points to the same location; the integer at that location
12       cannot be modified */
13    const int *const ptr = &x;
14
15    printf( "%d\n", *ptr );
16
17    *ptr = 7; /* error: *ptr is const; cannot assign new value */
18    ptr = &y; /* error: ptr is const; cannot assign new address */
19    return 0; /* indicates successful termination */
20 } /* end main */
```

ptr is a constant pointer to an integer constant

error: *ptr is const; cannot assign new value
error: ptr is const; cannot assign new address

```
fig07_14.c: In function ‘main’:
fig07_14.c:17: error: assignment of read-only location
fig07_14.c:18: error: assignment of read-only variable ‘ptr’
```

Bubble Sort Using Call-by-Reference

- Let's improve the bubble sort program of Fig. 6.15 to use two functions—bubbleSort and swap.
  - Function bubbleSort sorts the array.
  - Function swap exchanges the array elements **`array[j]`** and **`array[j + 1]`**

Bubble Sort Using Call-by-Reference (Cont.)

- Example: fig07_15.c

```
 7 void bubbleSort( int * const array, const int size ); /* prototype */
 8
 9 int main( void )
10 {
11    /* initialize array a */
12    int a[ SIZE ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
13
14    int i; /* counter */
15
16    printf( "Data items in original order\n" );
17
18    /* loop through array a */
19    for ( i = 0; i < SIZE; i++ ) {
20       printf( "%4d", a[ i ] );
21    } /* end for */
22
23    bubbleSort( a, SIZE ); /* sort the array */
24
25    printf( "\nData items in ascending order\n" );
26
27    /* loop through array a */
28    for ( i = 0; i < SIZE; i++ ) {
29       printf( "%4d", a[ i ] );···
30    } /* end for */
31
32    printf( "\n" );
33    return 0; /* indicates successful termination */
34 } /* end main */
```

Bubble Sort Using Call-by-Reference (Cont.)

- Example: fig07_15.c

```
 7 void bubbleSort( int * const array, const int size ); /* prototype */
 8
 9 int main( void )
10 {
11    /* initialize array a */
12    int a[ SIZE ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
13
14    int i; /* counter */
15
16    printf( "Data items in original order\n" );
17
18    /* loop through array a */
19    for ( i = 0; i < SIZE; i++ ) {
20       printf( "%4d", a[ i ] );
21    } /* end for */
22
23    bubbleSort( a, SIZE ); /* sort the array */
24
25    printf( "\nData items in ascending order\n" );
26
27    /* loop through array a */
28    for ( i = 0; i < SIZE; i++ ) {
29       printf( "%4d", a[ i ] );···
30    } /* end for */
31
32    printf( "\n" );
33    return 0; /* indicates successful termination */
34 } /* end main */
```

Bubble Sort Using Call-by-Reference (Cont.)

- Example: fig07_15.c

```c
7  void bubbleSort( int * const array, const int size ); /* prototype */
8
9  int main( void )
10 {
11    /* initialize array a */
12    int a[ SIZE ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
13
14    int i; /* counter */
15
16    printf( "Data items in original order\n" );
17
18    /* loop through array a */
19    for ( i = 0; i < SIZE; i++ ) {
20       printf( "%4d", a[ i ] );
21    } /* end for */
22
23    bubbleSort( a, SIZE ); /* sort the array */
24
25    printf( "\nData items in ascending order\n" );
26
27    /* loop through array a */
28    for ( i = 0; i < SIZE; i++ ) {
29       printf( "%4d", a[ i ] );···
30    } /* end for */
31
32    printf( "\n" );
33    return 0; /* indicates successful termination */
34 } /* end main */
```

array is a constant pointer to an integer

Bubble Sort Using Call-by-Reference (Cont.)

- Example: fig07_15.c

```c
7  void bubbleSort( int * const array, const int size ); /* prototype */
8
9  int main( void )
10 {
11    /* initialize array a */
12    int a[ SIZE ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
13
14    int i; /* counter */
15
16    printf( "Data items in original order\n" );
17
18    /* loop through array a */
19    for ( i = 0; i < SIZE; i++ ) {
20       printf( "%4d", a[ i ] );
21    } /* end for */
22
23    bubbleSort( a, SIZE ); /* sort the array */
24
25    printf( "\nData items in ascending order\n" );
26
27    /* loop through array a */
28    for ( i = 0; i < SIZE; i++ ) {
29       printf( "%4d", a[ i ] );···
30    } /* end for */
31
32    printf( "\n" );
33    return 0; /* indicates successful termination */
34 } /* end main */
```

array is a constant pointer to an integer

Bubble Sort Using Call-by-Reference (Cont.)

- Example: fig07_15.c

```
 7 void bubbleSort( int * const array, const int size ); /* prototype */
 8
 9 int main( void )
10 {
11    /* initialize array a */
12    int a[ SIZE ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
13
14    int i; /* counter */
15
16    printf( "Data items in original order\n" );
17
18    /* loop through array a */
19    for ( i = 0; i < SIZE; i++ ) {
20       printf( "%4d", a[ i ] );
21    } /* end for */
22
23    bubbleSort( a, SIZE ); /* sort the array */
24
25    printf( "\nData items in ascending order\n" );
26
27    /* loop through array a */
28    for ( i = 0; i < SIZE; i++ ) {
29       printf( "%4d", a[ i ] );···
30    } /* end for */
31
32    printf( "\n" );
33    return 0; /* indicates successful termination */
34 } /* end main */
```

array is a constant pointer to an integer

invoke bubbleSort( )

Bubble Sort Using Call-by-Reference (Cont.)

- Example: fig07_15.c

```
37 void bubbleSort( int * const array, const int size )
38 {
39     void swap( int *element1Ptr, int *element2Ptr ); /* prototype */
40     int pass; /* pass counter */
41     int j; /* comparison counter */
42
43     /* loop to control passes */
44     for ( pass = 0; pass < size - 1; pass++ ) {
45
46         /* loop to control comparisons during each pass */
47         for ( j = 0; j < size - 1; j++ ) {
48
49             /* swap adjacent elements if they are out of order */
50             if ( array[ j ] > array[ j + 1 ] ) {
51                 swap( &array[ j ], &array[ j + 1 ] );
52             } /* end if */
53         } /* end inner for */
54     } /* end outer for */
55 } /* end function bubbleSort */
```

Bubble Sort Using Call-by-Reference (Cont.)

- Example: fig07_15.c

```
37 void bubbleSort( int * const array, const int size )
38 {
39    void swap( int *element1Ptr, int *element2Ptr ); /* prototype */
40    int pass; /* pass counter */
41    int j; /* comparison counter */
42
43    /* loop to control passes */
44    for ( pass = 0; pass < size - 1; pass++ ) {
45
46       /* loop to control comparisons during each pass */
47       for ( j = 0; j < size - 1; j++ ) {
48
49          /* swap adjacent elements if they are out of order */
50          if ( array[ j ] > array[ j + 1 ] ) {
51             swap( &array[ j ], &array[ j + 1 ] );
52          } /* end if */
53       } /* end inner for */
54    } /* end outer for */
55 } /* end function bubbleSort */
```

Bubble Sort Using Call-by-Reference (Cont.)

- Example: fig07_15.c

```
37 void bubbleSort( int * const array, const int size )
38 {
39    void swap( int *element1Ptr, int *element2Ptr ); /* prototype */
40    int pass; /* pass counter */
41    int j; /* comparison counter */
42
43    /* loop to control passes */
44    for ( pass = 0; pass < size - 1; pass++ ) {
45
46       /* loop to control comparisons during each pass */
47       for ( j = 0; j < size - 1; j++ ) {
48
49          /* swap adjacent elements if they are out of order */
50          if ( array[ j ] > array[ j + 1 ] ) {
51             swap( &array[ j ], &array[ j + 1 ] );
52          } /* end if */
53       } /* end inner for */
54    } /* end outer for */
55 } /* end function bubbleSort */
```

Bubble Sort Using Call-by-Reference (Cont.)

- Example: fig07_15.c

```
37 void bubbleSort( int * const array, const int size )
38 {
39    void swap( int *element1Ptr, int *element2Ptr ); /* prototype */
40    int pass; /* pass counter */
41    int j; /* comparison counter */
42
43    /* loop to control passes */
44    for ( pass = 0; pass < size - 1; pass++ ) {
45
46       /* loop to control comparisons during each pass */
47       for ( j = 0; j < size - 1; j++ ) {
48
49          /* swap adjacent elements if they are out of order */
50          if ( array[ j ] > array[ j + 1 ] ) {
51             swap( &array[ j ], &array[ j + 1 ] );
52          } /* end if */
53       } /* end inner for */
54    } /* end outer for */
55 } /* end function bubbleSort */
```

use call-by-reference to swap these two elements

Bubble Sort Using Call-by-Reference (Cont.)

- Example: fig07_15.c

```
37 void bubbleSort( int * const array, const int size )
38 {
39    void swap( int *element1Ptr, int *element2Ptr ); /* prototype */
40    int pass; /* pass counter */
41    int j; /* comparison counter */
42
43    /* loop to control passes */
44    for ( pass = 0; pass < size - 1; pass++ ) {
45
46       /* loop to control comparisons during each pass */
47       for ( j = 0; j < size - 1; j++ ) {
48
49          /* swap adjacent elements if they are out of order */
50          if ( array[ j ] > array[ j + 1 ] ) {
51             swap( &array[ j ], &array[ j + 1 ] );
52          } /* end if */
53       } /* end inner for */
54    } /* end outer for */
55 } /* end function bubbleSort */
```

use call-by-reference to swap these two elements

pass the addresses of two elements to swap

Bubble Sort Using Call-by-Reference (Cont.)

- Example: fig07_15.c

```
59 void swap( int *element1Ptr, int *element2Ptr )
60 {
61    int hold = *element1Ptr;
62    *element1Ptr = *element2Ptr;
63    *element2Ptr = hold;
64 } /* end function swap */
```

Bubble Sort Using Call-by-Reference (Cont.)

- Example: fig07_15.c

```
59 void swap( int *element1Ptr, int *element2Ptr )
60 {
61    int hold = *element1Ptr;
62    *element1Ptr = *element2Ptr;
63    *element2Ptr = hold;
64 } /* end function swap */
```

Bubble Sort Using Call-by-Reference (Cont.)

- Example: fig07_15.c

```
59 void swap( int *element1Ptr, int *element2Ptr )
60 {
61    int hold = *element1Ptr;
62    *element1Ptr = *element2Ptr;
63    *element2Ptr = hold;
64 } /* end function swap */
```

use a hold to put the temporary data

Bubble Sort Using Call-by-Reference (Cont.)

- Example: fig07_15.c

```
59 void swap( int *element1Ptr, int *element2Ptr )
60 {
61    int hold = *element1Ptr;
62    *element1Ptr = *element2Ptr;
63    *element2Ptr = hold;
64 } /* end function swap */
```

use a hold to put the temporary data

```
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
```

`sizeof` Operator

- C provides the special unary operator **`sizeof`** to determine the size in bytes of an array (or any other data type) during program compilation.

`sizeof` Operator (Cont.)

- Example: fig07_16.c

```
 6 size_t getSize( float *ptr ); /* prototype */
 7
 8 int main( void )
 9 {
10    float array[ 20 ]; /* create array */
11
12    printf( "The number of bytes in the array is %d"
13            "\nThe number of bytes returned by getSize is %d\n",
14            sizeof( array ), getSize( array ) );
15    return 0; /* indicates successful termination */
16 } /* end main */
17
18 /* return size of ptr */
19 size_t getSize( float *ptr )
20 {
21    return sizeof( ptr );
22 } /* end function getSize */
```

`sizeof` Operator (Cont.)

- Example: fig07_16.c

```
6  size_t getSize( float *ptr ); /* prototype */
7
8  int main( void )
9  {
10     float array[ 20 ]; /* create array */
11
12     printf( "The number of bytes in the array is %d"
13             "\nThe number of bytes returned by getSize is %d\n",
14             sizeof( array ), getSize( array ) );
15     return 0; /* indicates successful termination */
16 } /* end main */
17
18 /* return size of ptr */
19 size_t getSize( float *ptr )
20 {
21     return sizeof( ptr );
22 } /* end function getSize */
```

`sizeof` Operator (Cont.)

- Example: fig07_16.c

```
 6 size_t getSize( float *ptr ); /* prototype */
 7
 8 int main( void )
 9 {
10    float array[ 20 ]; /* create array */
11
12    printf( "The number of bytes in the array is %d"
13            "\nThe number of bytes returned by getSize is %d\n",
14            sizeof( array ), getSize( array ) );
15    return 0; /* indicates successful termination */
16 } /* end main */
17
18 /* return size of ptr */
19 size_t getSize( float *ptr )
20 {
21    return sizeof( ptr );
22 } /* end function getSize */
```

size_t is a type defined by the C standard as the integral type of the value returned by operator sizeof

`sizeof` Operator (Cont.)

- Example: fig07_16.c

```
 6 size_t getSize( float *ptr ); /* prototype */
 7
 8 int main( void )
 9 {
10    float array[ 20 ]; /* create array */
11
12    printf( "The number of bytes in the array is %d"
13            "\nThe number of bytes returned by getSize is %d\n",
14            sizeof( array ), getSize( array ) );
15    return 0; /* indicates successful termination */
16 } /* end main */
17
18 /* return size of ptr */
19 size_t getSize( float *ptr )
20 {
21    return sizeof( ptr );
22 } /* end function getSize */
```

size_t is a type defined by the C standard as the integral type of the value returned by operator sizeof

`sizeof` Operator (Cont.)

- Example: fig07_16.c

```
 6 size_t getSize( float *ptr ); /* prototype */
 7
 8 int main( void )
 9 {
10    float array[ 20 ]; /* create array */
11
12    printf( "The number of bytes in the array is %d"
13            "\nThe number of bytes returned by getSize is %d\n",
14            sizeof( array ), getSize( array ) );
15    return 0; /* indicates successful termination */
16 } /* end main */
17
18 /* return size of ptr */
19 size_t getSize( float *ptr )
20 {
21    return sizeof( ptr );
22 } /* end function getSize */
```

size_t is a type defined by the C standard as the integral type of the value returned by operator sizeof

return the size of the array pointed by **ptr**

`sizeof` Operator (Cont.)

- Example: fig07_16.c

```
 6 size_t getSize( float *ptr ); /* prototype */
 7
 8 int main( void )
 9 {
10    float array[ 20 ]; /* create array */
11
12    printf( "The number of bytes in the array is %d"
13            "\nThe number of bytes returned by getSize is %d\n",
14            sizeof( array ), getSize( array ) );
15    return 0; /* indicates successful termination */
16 } /* end main */
17
18 /* return size of ptr */
19 size_t getSize( float *ptr )
20 {
21    return sizeof( ptr );
22 } /* end function getSize */
```

size_t is a type defined by the C standard as the integral type of the value returned by operator sizeof

return the size of the array pointed by **ptr**

```
The number of bytes in the array is 80
The number of bytes returned by getSize is 8
```

`sizeof` Operator (Cont.)

- The number of elements in an array also can be determined with **`sizeof`**.
- For example, consider the following array definition:

```
double real[22];
```

- To determine the number of elements in the array, the following expression can be used:

```
sizeof(real)/sizeof(real[0])
```

`sizeof` Operator (Cont.)

- Type **`size_t`** is a type defined by the C standard as the integral type (unsigned or unsigned long) of the value returned by operator sizeof.
- Type **`size_t`** is defined in header **`<stddef.h>`** (which is included by several headers, such as **`<stdio.h>`**).
- [Note: If you attempt to compile Fig. 7.16 and receive errors, simply include **`<stddef.h>`** in your program.]

`sizeof` Operator (Cont.)

- Example: fig07_17.c

```
7     char c;···········
8     short s;·········
9     int i;·······
10    long l;·········
11    float f;········
12    double d;·········
13    long double ld;···
14    int array[ 20 ]; /* create array of 20 int elements */
15    int *ptr = array; /* create pointer to array */
16
17    printf( "     sizeof c = %d\tsizeof(char)  = %d"···
18            "\n     sizeof s = %d\tsizeof(short) = %d"···
19            "\n     sizeof i = %d\tsizeof(int) = %d"···
20            "\n     sizeof l = %d\tsizeof(long) = %d"···
21            "\n     sizeof f = %d\tsizeof(float) = %d"···
22            "\n     sizeof d = %d\tsizeof(double) = %d"···
23            "\n    sizeof ld = %d\tsizeof(long double) = %d"···
24            "\n sizeof array = %d"···
25            "\n   sizeof ptr = %d\n",····
26           sizeof c, sizeof( char ), sizeof s, sizeof( short ), sizeof i,
27           sizeof( int ), sizeof l, sizeof( long ), sizeof f,·
28           sizeof( float ), sizeof d, sizeof( double ), sizeof ld,·
29           sizeof( long double ), sizeof array, sizeof ptr );··
```

```
    sizeof c = 1      sizeof(char)  = 1
    sizeof s = 2      sizeof(short) = 2
    sizeof i = 4      sizeof(int) = 4
    sizeof l = 8      sizeof(long) = 8
    sizeof f = 4      sizeof(float) = 4
    sizeof d = 8      sizeof(double) = 8
   sizeof ld = 16     sizeof(long double) = 16
sizeof array = 80
  sizeof ptr = 8
```

`sizeof` Operator (Cont.)

- Operator **`sizeof`** can be applied to any **variable name**, **type** or **value** (including the value of an expression).
- When applied to a variable name (that is not an array name) or a constant, the number of bytes used to store the specific type of variable or constant is returned.
- The **parentheses** used with sizeof are required if a **type name with two words** is supplied as its operand (such as long double or unsigned short).

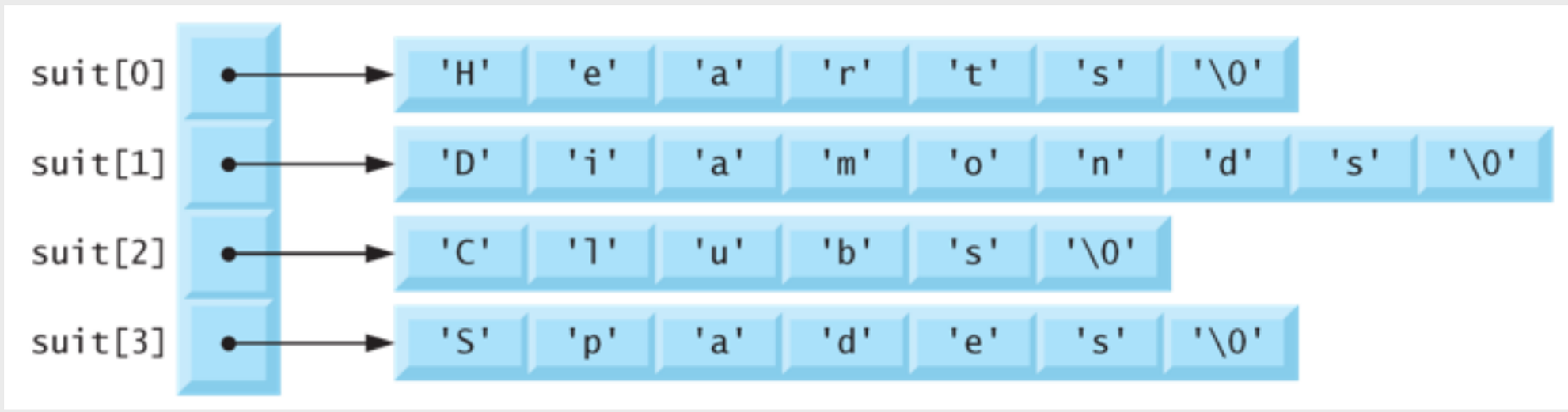
Arrays of Pointers

- Arrays may contain pointers.
- A common use of an **array of pointers** is to form an **array of strings**, referred to simply as a **string array**.
- Consider the definition of string array suit, which might be useful in representing a deck of cards.

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

Arrays of Pointers (Cont.)

- Each is stored in memory as a null-terminated character string that is one character longer than the number of characters between quotes.
- The four strings are 7, 9, 6 and 7 characters long, respectively.



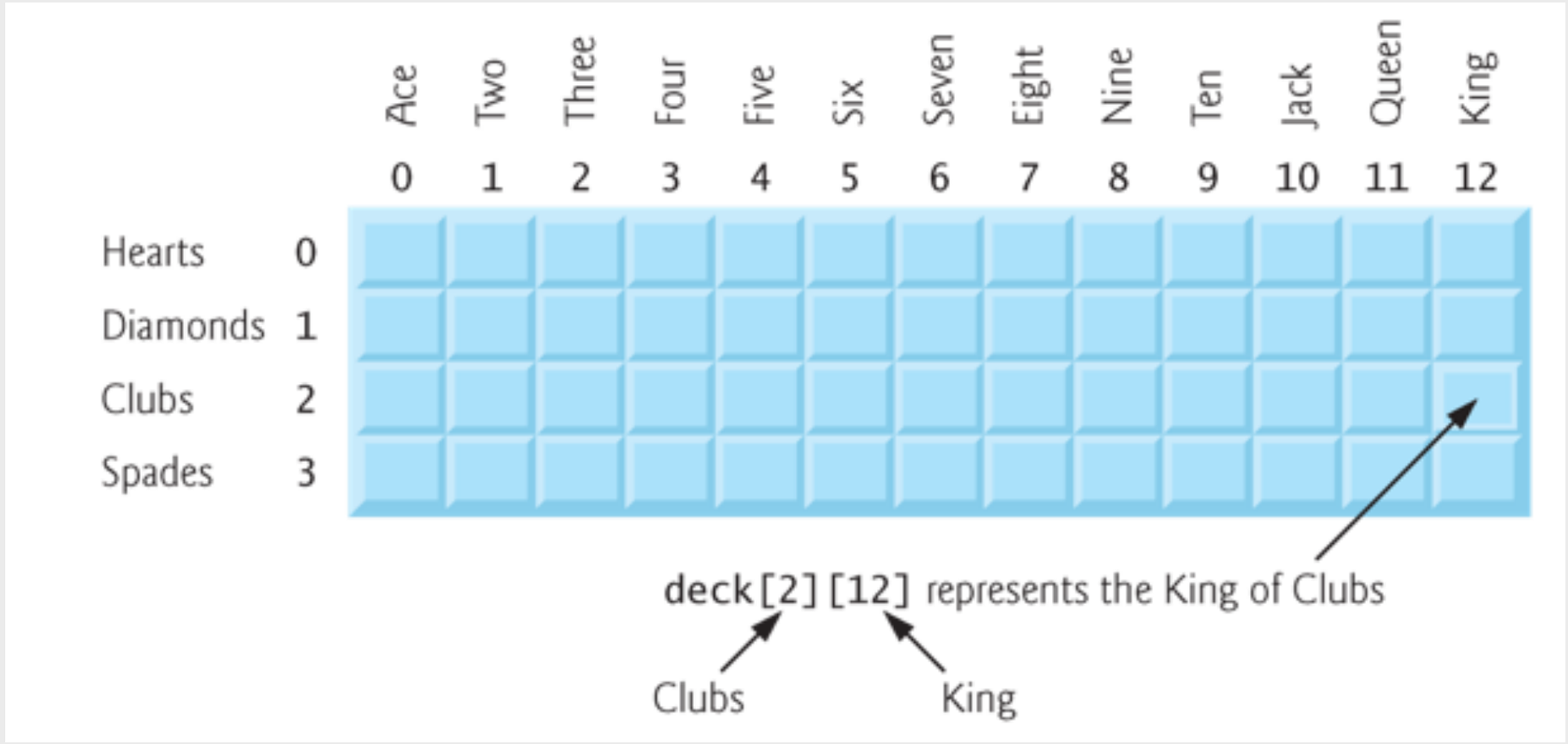
Arrays of Pointers (Cont.)

- Each pointer points to the first character of its corresponding string.
- Such a data structure would have to have a fixed number of columns per row, and that number would have to be as large as the largest string.
- Therefore, considerable memory could be wasted when a large number of strings were being stored with most strings shorter than the longest string.

Case Study: Card Shuffling and Dealing Simulation

- Using the top-down, stepwise refinement approach, we develop a program that will shuffle a deck of 52 playing cards and then deal each of the 52 cards.
- We use 4-by-13 double-subscripted array deck to represent the deck of playing cards.
  - The rows correspond to the suits—row 0 corresponds to hearts, row 1 to diamonds, row 2 to clubs and row 3 to spades.
  - The columns correspond to the face values of the cards—0 through 9 correspond to ace through ten, and columns 10 through 12 correspond to jack, queen and king.

Case Study: Card Shuffling and Dealing Simulation (Cont.)



Case Study: Card Shuffling and Dealing Simulation (Cont.)

- Example: fig07_24.c

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

int main( void )
{
    /* 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 };

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

    shuffle( deck ); /* shuffle the deck */
    deal( deck, face, suit ); /* deal the deck */
    return 0; /* indicates successful termination */
} /* end main */
```

Case Study: Card Shuffling and Dealing Simulation (Cont.)

- Example: fig07_24.c

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

int main( void )
{
    /* 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 };

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

    shuffle( deck ); /* shuffle the deck */
    deal( deck, face, suit ); /* deal the deck */
    return 0; /* indicates successful termination */
} /* end main */
```

Case Study: Card Shuffling and Dealing Simulation (Cont.)

- Example: fig07_24.c

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

int main( void )
{
    /* 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 };

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

    shuffle( deck ); /* shuffle the deck */
    deal( deck, face, suit ); /* deal the deck */
    return 0; /* indicates successful termination */
} /* end main */
```

four patterns

Case Study: Card Shuffling and Dealing Simulation (Cont.)

- Example: fig07_24.c

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

int main( void )
{
    /* 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 };

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

    shuffle( deck ); /* shuffle the deck */
    deal( deck, face, suit ); /* deal the deck */
    return 0; /* indicates successful termination */
} /* end main */
```

four patterns

Case Study: Card Shuffling and Dealing Simulation (Cont.)

- Example: fig07_24.c

```
 8 void shuffle( int wDeck[][ 13 ] );
 9 void deal( const int wDeck[][ 13 ], const char *wFace[], const char *wSuit[] );
10
11 int main( void )
12 {
13    /* initialize suit array */
14    const char *suit[ 4 ] = { "Hearts", "Diamonds", "Clubs", "Spades" };
15
16    /* initialize face array */
17    const char *face[ 13 ] =
18    { "Ace", "Deuce", "Three", "Four",
19       "Five", "Six", "Seven", "Eight",
20       "Nine", "Ten", "Jack", "Queen", "King" };
21
22    /* initialize deck array */
23    int deck[ 4 ][ 13 ] = { 0 };
24
25    srand( time( 0 ) ); /* seed random-number generator */
26
27    shuffle( deck ); /* shuffle the deck */
28    deal( deck, face, suit ); /* deal the deck */
29    return 0; /* indicates successful termination */
30 } /* end main */
```

four patterns

13 numbers

Case Study: Card Shuffling and Dealing Simulation (Cont.)

- Example: fig07_24.c

```
 8 void shuffle( int wDeck[][ 13 ] );
 9 void deal( const int wDeck[][ 13 ], const char *wFace[], const char *wSuit[] );
10
11 int main( void )
12 {
13    /* initialize suit array */
14    const char *suit[ 4 ] = { "Hearts", "Diamonds", "Clubs", "Spades" };
15
16    /* initialize face array */
17    const char *face[ 13 ] =
18    { "Ace", "Deuce", "Three", "Four",
19       "Five", "Six", "Seven", "Eight",
20       "Nine", "Ten", "Jack", "Queen", "King" };
21
22    /* initialize deck array */
23    int deck[ 4 ][ 13 ] = { 0 };
24
25    srand( time( 0 ) ); /* seed random-number generator */
26
27    shuffle( deck ); /* shuffle the deck */
28    deal( deck, face, suit ); /* deal the deck */
29    return 0; /* indicates successful termination */
30 } /* end main */
```

four patterns

13 numbers

Case Study: Card Shuffling and Dealing Simulation (Cont.)

- Example: fig07_24.c

```
 8 void shuffle( int wDeck[][ 13 ] );
 9 void deal( const int wDeck[][ 13 ], const char *wFace[], const char *wSuit[] );
10
11 int main( void )
12 {
13    /* initialize suit array */
14    const char *suit[ 4 ] = { "Hearts", "Diamonds", "Clubs", "Spades" };
15
16    /* initialize face array */
17    const char *face[ 13 ] =
18    { "Ace", "Deuce", "Three", "Four",
19       "Five", "Six", "Seven", "Eight",
20       "Nine", "Ten", "Jack", "Queen", "King" };
21
22    /* initialize deck array */
23    int deck[ 4 ][ 13 ] = { 0 };
24
25    srand( time( 0 ) ); /* seed random-number generator */
26
27    shuffle( deck ); /* shuffle the deck */
28    deal( deck, face, suit ); /* deal the deck */
29    return 0; /* indicates successful termination */
30 } /* end main */
```

four patterns

13 numbers

initialize a deck

Case Study: Card Shuffling and Dealing Simulation (Cont.)

- Example: fig07_24.c

```c
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 */

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

Case Study: Card Shuffling and Dealing Simulation (Cont.)

- Example: fig07_24.c

```
33 void shuffle( int wDeck[][ 13 ] )
34 {
35     int row; /* row number */
36     int column; /* column number */
37     int card; /* counter */
38
39     /* for each of the 52 cards, choose slot of deck randomly */
40     for ( card = 1; card <= 52; card++ ) {
41
42         /* choose new random location until unoccupied slot found */
43         do {
44             row = rand() % 4;
45             column = rand() % 13;
46         } while( wDeck[ row ][ column ] != 0 ); /* end do...while */
47
48         /* place card number in chosen slot of deck */
49         wDeck[ row ][ column ] = card;
50     } /* end for */
51 } /* end function shuffle */
```

Case Study: Card Shuffling and Dealing Simulation (Cont.)

- Example: fig07_24.c

```
33 void shuffle( int wDeck[][ 13 ] )
34 {
35     int row; /* row number */
36     int column; /* column number */
37     int card; /* counter */
38
39     /* for each of the 52 cards, choose slot of deck randomly */
40     for ( card = 1; card <= 52; card++ ) {
41
42         /* choose new random location until unoccupied slot found */
43         do {
44             row = rand() % 4;
45             column = rand() % 13;
46         } while( wDeck[ row ][ column ] != 0 ); /* end do...while */
47
48         /* place card number in chosen slot of deck */
49         wDeck[ row ][ column ] = card;
50     } /* end for */
51 } /* end function shuffle */
```

shuffle the deck

Case Study: Card Shuffling and Dealing Simulation (Cont.)

- Example: fig07_24.c

```c
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 ) {
                    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 */
```

Case Study: Card Shuffling and Dealing Simulation (Cont.)

- Example: fig07_24.c

```
54 void deal( const int wDeck[][ 13 ], const char *wFace[], const char *wSuit[] )
55 {
56    int card; /* card counter */
57    int row; /* row counter */
58    int column; /* column counter */
59
60    /* deal each of the 52 cards */
61    for ( card = 1; card <= 52; card++ ) {
62
63       /* loop through rows of wDeck */
64       for ( row = 0; row <= 3; row++ ) {
65
66          /* loop through columns of wDeck for current row */
67          for ( column = 0; column <= 12; column++ ) {
68
69             /* if slot contains current card, display card */
70             if ( wDeck[ row ][ column ] == card ) {
71                printf( "%5s of %-8s%c", wFace[ column ], wSuit[ row ],
72                   card % 2 == 0 ? '\n' : '\t' );
73             } /* end if */
74          } /* end for */
75       } /* end for */
76    } /* end for */
77 } /* end function deal */
```

Case Study: Card Shuffling and Dealing Simulation (Cont.)

- Example: fig07_24.c

```
54 void deal( const int wDeck[][ 13 ], const char *wFace[], const char *wSuit[] )
55 {
56    int card; /* card counter */
57    int row; /* row counter */
58    int column; /* column counter */
59
60    /* deal each of the 52 cards */
61    for ( card = 1; card <= 52; card++ ) {
62
63       /* loop through rows of wDeck */
64       for ( row = 0; row <= 3; row++ ) {
65
66          /* loop through columns of wDeck for current row */
67          for ( column = 0; column <= 12; column++ ) {
68
69             /* if slot contains current card, display card */
70             if ( wDeck[ row ][ column ] == card ) {
71                printf( "%5s of %-8s%c", wFace[ column ], wSuit[ row ],
72                   card % 2 == 0 ? '\n' : '\t' );
73             } /* end if */
74          } /* end for */
75       } /* end for */
76    } /* end for */
77 } /* end function deal */
```

search for each slot, and if slot contains current card, display card

Case Study: Card Shuffling and Dealing Simulation (Cont.)

- Example: fig07_24.c

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

Case Study: Card Shuffling and Dealing Simulation (Cont.)

- There's a weakness in the dealing algorithm.
- Once a match is found, the two inner **`for`** statements continue searching the remaining elements of **deck** for a match.
- We correct this deficiency in this chapter's exercises and in a Chapter 10 case study.

Pointers to Functions

- A pointer to a function contains the address of the function in memory.
- A function name is really the starting address in memory of the code that performs the function's task.
- Pointers to functions can be passed to functions, returned from functions, stored in arrays and assigned to other function pointers.

Pointers to Functions (Cont.)

- Example: fig07_26.c

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

```
17    int a[ SIZE ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
18
19    printf( "Enter 1 to sort in ascending order,\n"
20            "Enter 2 to sort in descending order: " );
21    scanf( "%d",  &order );
22
23    printf( "\nData items in original order\n" );
24
25    /* output original array */
26    for ( counter = 0; counter < SIZE; counter++ ) {
27       printf( "%5d", a[ counter ] );
28    } /* end for */
29
30    /* sort array in ascending order; pass function ascending as an
31       argument to specify ascending sorting order */
32    if ( order == 1 ) {
33       bubble( a, SIZE, ascending );
34       printf( "\nData items in ascending order\n" );
35    } /* end if */.
36    else { /* pass function descending */
37       bubble( a, SIZE, descending );
38       printf( "\nData items in descending order\n" );
39    } /* end else */
```

Pointers to Functions (Cont.)

- Example: fig07_26.c

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

```
17    int a[ SIZE ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
18
19    printf( "Enter 1 to sort in ascending order,\n"
20            "Enter 2 to sort in descending order: " );
21    scanf( "%d",  &order );
22
23    printf( "\nData items in original order\n" );
24
25    /* output original array */
26    for ( counter = 0; counter < SIZE; counter++ ) {
27       printf( "%5d", a[ counter ] );
28    } /* end for */
29
30    /* sort array in ascending order; pass function ascending as an
31       argument to specify ascending sorting order */
32    if ( order == 1 ) {
33       bubble( a, SIZE, ascending );
34       printf( "\nData items in ascending order\n" );
35    } /* end if */.
36    else { /* pass function descending */
37       bubble( a, SIZE, descending );
38       printf( "\nData items in descending order\n" );
39    } /* end else */
```

Pointers to Functions (Cont.)

- Example: fig07_26.c

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

function pointer

```
17    int a[ SIZE ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
18
19    printf( "Enter 1 to sort in ascending order,\n"
20            "Enter 2 to sort in descending order: " );
21    scanf( "%d",  &order );
22
23    printf( "\nData items in original order\n" );
24
25    /* output original array */
26    for ( counter = 0; counter < SIZE; counter++ ) {
27       printf( "%5d", a[ counter ] );
28    } /* end for */
29
30    /* sort array in ascending order; pass function ascending as an
31       argument to specify ascending sorting order */
32    if ( order == 1 ) {
33       bubble( a, SIZE, ascending );
34       printf( "\nData items in ascending order\n" );
35    } /* end if */.
36    else { /* pass function descending */
37       bubble( a, SIZE, descending );
38       printf( "\nData items in descending order\n" );
39    } /* end else */
```

Pointers to Functions (Cont.)

- Example: fig07_26.c

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

function pointer

```
17    int a[ SIZE ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
18
19    printf( "Enter 1 to sort in ascending order,\n"
20            "Enter 2 to sort in descending order: " );
21    scanf( "%d",  &order );
22
23    printf( "\nData items in original order\n" );
24
25    /* output original array */
26    for ( counter = 0; counter < SIZE; counter++ ) {
27       printf( "%5d", a[ counter ] );
28    } /* end for */
29
30    /* sort array in ascending order; pass function ascending as an
31       argument to specify ascending sorting order */
32    if ( order == 1 ) {
33       bubble( a, SIZE, ascending );
34       printf( "\nData items in ascending order\n" );
35    } /* end if */.
36    else { /* pass function descending */
37       bubble( a, SIZE, descending );
38       printf( "\nData items in descending order\n" );
39    } /* end else */
```

Pointers to Functions (Cont.)

- Example: fig07_26.c

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

function pointer

```
17    int a[ SIZE ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
18
19    printf( "Enter 1 to sort in ascending order,\n"
20            "Enter 2 to sort in descending order: " );
21    scanf( "%d",  &order );
22
23    printf( "\nData items in original order\n" );
24
25    /* output original array */
26    for ( counter = 0; counter < SIZE; counter++ ) {
27       printf( "%5d", a[ counter ] );
28    } /* end for */
29
30    /* sort array in ascending order; pass function ascending as an
31       argument to specify ascending sorting order */
32    if ( order == 1 ) {
33       bubble( a, SIZE, ascending );
34       printf( "\nData items in ascending order\n" );
35    } /* end if */.
36    else { /* pass function descending */
37       bubble( a, SIZE, descending );
38       printf( "\nData items in descending order\n" );
39    } /* end else */
```

1 for ascending;
2 for descending

Pointers to Functions (Cont.)

- Example: fig07_26.c

```
53 void bubble( int work[], const int size, int (*compare)( int a, int b ) )
54 {
55     int pass; /* pass counter */
56     int count; /* comparison counter */
57
58     void swap( int *element1Ptr, int *element2ptr ); /* prototype */
59
60     /* loop to control passes */
61     for ( pass = 1; pass < size; pass++ ) {
62
63         /* loop to control number of comparisons per pass */
64         for ( count = 0; count < size - 1; count++ ) {
65
66             /* if adjacent elements are out of order, swap them */
67             if ( (*compare)( work[ count ], work[ count + 1 ] ) ) {
68                 swap( &work[ count ], &work[ count + 1 ] );
69             } /* end if */
70
71         } /* end for */
72
73     } /* end for */
74
75 } /* end function bubble */
```

Pointers to Functions (Cont.)

- Example: fig07_26.c

```
53 void bubble( int work[], const int size, int (*compare)( int a, int b ) )
54 {
55     int pass; /* pass counter */
56     int count; /* comparison counter */
57
58     void swap( int *element1Ptr, int *element2ptr ); /* prototype */
59
60     /* loop to control passes */
61     for ( pass = 1; pass < size; pass++ ) {
62
63         /* loop to control number of comparisons per pass */
64         for ( count = 0; count < size - 1; count++ ) {
65
66             /* if adjacent elements are out of order, swap them */
67             if ( (*compare)( work[ count ], work[ count + 1 ] ) ) {
68                 swap( &work[ count ], &work[ count + 1 ] );
69             } /* end if */
70
71         } /* end for */
72
73     } /* end for */
74
75 } /* end function bubble */
```

Pointers to Functions (Cont.)

- Example: fig07_26.c

```
53 void bubble( int work[], const int size, int (*compare)( int a, int b ) )
54 {
55     int pass; /* pass counter */
56     int count; /* comparison counter */
57
58     void swap( int *element1Ptr, int *element2ptr ); /* prototype */
59
60     /* loop to control passes */
61     for ( pass = 1; pass < size; pass++ ) {
62
63         /* loop to control number of comparisons per pass */
64         for ( count = 0; count < size - 1; count++ ) {
65
66             /* if adjacent elements are out of order, swap them */
67             if ( (*compare)( work[ count ], work[ count + 1 ] ) ) {
68                 swap( &work[ count ], &work[ count + 1 ] );
69             } /* end if */
70
71         } /* end for */
72
73     } /* end for */
74
75 } /* end function bubble */
```

function pointer

Pointers to Functions (Cont.)

- Example: fig07_26.c

```
53 void bubble( int work[], const int size, int (*compare)( int a, int b ) )
54 {
55     int pass; /* pass counter */
56     int count; /* comparison counter */
57
58     void swap( int *element1Ptr, int *element2ptr ); /* prototype */
59
60     /* loop to control passes */
61     for ( pass = 1; pass < size; pass++ ) {
62
63         /* loop to control number of comparisons per pass */
64         for ( count = 0; count < size - 1; count++ ) {
65
66             /* if adjacent elements are out of order, swap them */
67             if ( (*compare)( work[ count ], work[ count + 1 ] ) ) {
68                 swap( &work[ count ], &work[ count + 1 ] );
69             } /* end if */
70
71         } /* end for */
72
73     } /* end for */
74
75 } /* end function bubble */
```

function pointer

Pointers to Functions (Cont.)

- Example: fig07_26.c

```
53 void bubble( int work[], const int size, int (*compare)( int a, int b ) )
54 {
55     int pass; /* pass counter */
56     int count; /* comparison counter */
57
58     void swap( int *element1Ptr, int *element2ptr ); /* prototype */
59
60     /* loop to control passes */
61     for ( pass = 1; pass < size; pass++ ) {
62
63         /* loop to control number of comparisons per pass */
64         for ( count = 0; count < size - 1; count++ ) {
65
66             /* if adjacent elements are out of order, swap them */
67             if ( (*compare)( work[ count ], work[ count + 1 ] ) ) {
68                 swap( &work[ count ], &work[ count + 1 ] );
69             } /* end if */
70
71         } /* end for */
72
73     } /* end for */
74
75 } /* end function bubble */
```

function pointer

invoke the function via function pointer **(*compare)**

Pointers to Functions (Cont.)

- Example: fig07_26.c

```
79 void swap( int *element1Ptr, int *element2Ptr )
80 {
81     int hold; /* temporary holding variable */
82
83     hold = *element1Ptr;
84     *element1Ptr = *element2Ptr;
85     *element2Ptr = hold;
86 } /* end function swap */
```

```
90 int ascending( int a, int b )
91 {
92     return b < a; /* swap if b is less than a */
93
94 } /* end function ascending */
```

```
98 int descending( int a, int b )
99 {
100     return b > a; /* swap if b is greater than a */
101
102 } /* end function descending */
```

Pointers to Functions (Cont.)

- Example: fig07_26.c

```
79 void swap( int *element1Ptr, int *element2Ptr )
80 {
81    int hold; /* temporary holding variable */
82
83    hold = *element1Ptr;
84    *element1Ptr = *element2Ptr;
85    *element2Ptr = hold;
86 } /* end function swap */
```

```
90 int ascending( int a, int b )
91 {
92    return b < a; /* swap if b is less than a */
93
94 } /* end function ascending */
```

```
98 int descending( int a, int b )
99 {
100    return b > a; /* swap if b is greater than a */
101
102 } /* end function descending */
```

```
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
```

Pointers to Functions (Cont.)

- Example: fig07_26.c

```
79 void swap( int *element1Ptr, int *element2Ptr )
80 {
81    int hold; /* temporary holding variable */
82
83    hold = *element1Ptr;
84    *element1Ptr = *element2Ptr;
85    *element2Ptr = hold;
86 } /* end function swap */
```

```
90 int ascending( int a, int b )
91 {
92    return b < a; /* swap if b is less than a */
93
94 } /* end function ascending */
```

```
98 int descending( int a, int b )
99 {
100    return b > a; /* swap if b is greater than a */
101
102 } /* end function descending */
```

```
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
```

Pointers to Functions (Cont.)

- The following parameter appears in the function header for bubble

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

- This tells bubble to expect a parameter (* compare) that is a **pointer to a function** that receives two integer parameters and returns an integer result.

Pointers to Functions (Cont.)

- Parentheses are needed around **`*compare`** to group `*` with **`compare`** to indicate that compare is a pointer.
- If we had not included the parentheses, the declaration would have been

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

which declares a function that receives two integers as parameters and returns a pointer to an integer.

Pointers to Functions (Cont.)

- A common use of function pointers is in text-based **menu-driven systems**.
- A user is prompted to select an option from a menu (possibly from 1 to 5) by typing the menu item's number.
- The user's choice is used as a subscript in the array, and the pointer in the array is used to call the function.

Pointers to Functions (Cont.)

- Example: fig07_28.c

```c
/* prototypes */
void function1( int a );
void function2( int b );
void function3( int c );

int main( void )
{
   /* initialize array of 3 pointers to functions that each take an
      int argument and return void */
   void (*f[ 3 ])( int ) = { function1, function2, function3 };

   int choice; /* variable to hold user's choice */

   printf( "Enter a number between 0 and 2, 3 to end: " );
   scanf( "%d", &choice );

   /* process user's choice */
   while ( choice >= 0 && choice < 3 ) {

      /* invoke function at location choice in array f and pass
         choice as an argument */
      (*f[ choice ])( choice );

      printf( "Enter a number between 0 and 2, 3 to end: ");
      scanf( "%d", &choice );
   } /* end while */

   printf( "Program execution completed.\n" );
   return 0; /* indicates successful termination */
} /* end main */
```

Pointers to Functions (Cont.)

- Example: fig07_28.c

```
5  /* prototypes */
6  void function1( int a );
7  void function2( int b );
8  void function3( int c );
9
10 int main( void )
11 {
12     /* initialize array of 3 pointers to functions that each take an
13        int argument and return void */
14     void (*f[ 3 ])( int ) = { function1, function2, function3 };
15
16     int choice; /* variable to hold user's choice */
17
18     printf( "Enter a number between 0 and 2, 3 to end: " );
19     scanf( "%d", &choice );
20
21     /* process user's choice */
22     while ( choice >= 0 && choice < 3 ) {
23
24         /* invoke function at location choice in array f and pass
25            choice as an argument */
26         (*f[ choice ])( choice );
27
28         printf( "Enter a number between 0 and 2, 3 to end: ");
29         scanf( "%d", &choice );
30     } /* end while */
31
32     printf( "Program execution completed.\n" );
33     return 0; /* indicates successful termination */
34 } /* end main */
```

Pointers to Functions (Cont.)

- Example: fig07_28.c

```
5  /* prototypes */
6  void function1( int a );
7  void function2( int b );
8  void function3( int c );
9
10 int main( void )
11 {
12     /* initialize array of 3 pointers to functions that each take an
13        int argument and return void */
14     void (*f[ 3 ])( int ) = { function1, function2, function3 };
15
16     int choice; /* variable to hold user's choice */
17
18     printf( "Enter a number between 0 and 2, 3 to end: " );
19     scanf( "%d", &choice );
20
21     /* process user's choice */
22     while ( choice >= 0 && choice < 3 ) {
23
24        /* invoke function at location choice in array f and pass
25           choice as an argument */
26        (*f[ choice ])( choice );
27
28        printf( "Enter a number between 0 and 2, 3 to end: ");
29        scanf( "%d", &choice );
30     } /* end while */
31
32     printf( "Program execution completed.\n" );
33     return 0; /* indicates successful termination */
34 } /* end main */
```

an array of three pointers to functions

Pointers to Functions (Cont.)

- Example: fig07_28.c

```
5  /* prototypes */
6  void function1( int a );
7  void function2( int b );
8  void function3( int c );
9
10 int main( void )
11 {
12     /* initialize array of 3 pointers to functions that each take an
13        int argument and return void */
14     void (*f[ 3 ])( int ) = { function1, function2, function3 };
15
16     int choice; /* variable to hold user's choice */
17
18     printf( "Enter a number between 0 and 2, 3 to end: " );
19     scanf( "%d", &choice );
20
21     /* process user's choice */
22     while ( choice >= 0 && choice < 3 ) {
23
24        /* invoke function at location choice in array f and pass
25           choice as an argument */
26        (*f[ choice ])( choice );
27
28        printf( "Enter a number between 0 and 2, 3 to end: ");
29        scanf( "%d", &choice );
30     } /* end while */
31
32     printf( "Program execution completed.\n" );
33     return 0; /* indicates successful termination */
34 } /* end main */
```

an array of three pointers to functions

Pointers to Functions (Cont.)

- Example: fig07_28.c

```
5  /* prototypes */
6  void function1( int a );
7  void function2( int b );
8  void function3( int c );
9
10 int main( void )
11 {
12     /* initialize array of 3 pointers to functions that each take an
13        int argument and return void */
14     void (*f[ 3 ])( int ) = { function1, function2, function3 };
15
16     int choice; /* variable to hold user's choice */
17
18     printf( "Enter a number between 0 and 2, 3 to end: " );
19     scanf( "%d", &choice );
20
21     /* process user's choice */
22     while ( choice >= 0 && choice < 3 ) {
23
24        /* invoke function at location choice in array f and pass
25           choice as an argument */
26        (*f[ choice ])( choice );
27
28        printf( "Enter a number between 0 and 2, 3 to end: ");
29        scanf( "%d", &choice );
30     } /* end while */
31
32     printf( "Program execution completed.\n" );
33     return 0; /* indicates successful termination */
34 } /* end main */
```

an array of three pointers to functions

invoke the corresponding function

Pointers to Functions (Cont.)

- Example: fig07_28.c

```
36 void function1( int a )
37 {
38    printf( "You entered %d so function1 was called\n\n", a );
39 } /* end function1 */
40
41 void function2( int b )
42 {
43    printf( "You entered %d so function2 was called\n\n", b );
44 } /* end function2 */
45
46 void function3( int c )
47 {
48    printf( "You entered %d so function3 was called\n\n", c );
49 } /* end function3 */
```

Pointers to Functions (Cont.)

- Example: fig07_28.c

```c
void function1( int a )
{
   printf( "You entered %d so function1 was called\n\n", a );
} /* end function1 */

void function2( int b )
{
   printf( "You entered %d so function2 was called\n\n", b );
} /* end function2 */

void function3( int c )
{
   printf( "You entered %d so function3 was called\n\n", c );
} /* end function3 */
```

```
Enter a number between 0 and 2, 3 to end: 1
You entered 1 so function2 was called

Enter a number between 0 and 2, 3 to end: 0
You entered 0 so function1 was called

Enter a number between 0 and 2, 3 to end: 2
You entered 2 so function3 was called

Enter a number between 0 and 2, 3 to end: 3
Program execution completed.
```

Pointers to Functions (Cont.)

- The definition:

```
void (*f[3])(int) = {function1, function2,
function3}
```

"**f** is an array of 3 pointers to functions that each take an **int** as an argument and return **void**." The array is initialized with the names of the three functions.

- In the function call,

```
(*f[choice])(choice);
```

f[choice] selects the pointer at location **choice** in the array.

the pointer is dereferenced to call the function, and **choice** is passed as the argument to the function.