7

C Pointers

© 2007 Pearson Education, Inc. All rights reserved.

7.2 Pointer Variable Definitions and Initialization

- **Pointer variables**
  - **Contain memory addresses as their values**
  - **Normal variables contain a specific value (direct reference)**
  - **Pointers contain address of a variable that has a specific value (indirect reference)**
  - **Indirection – referencing a pointer value**

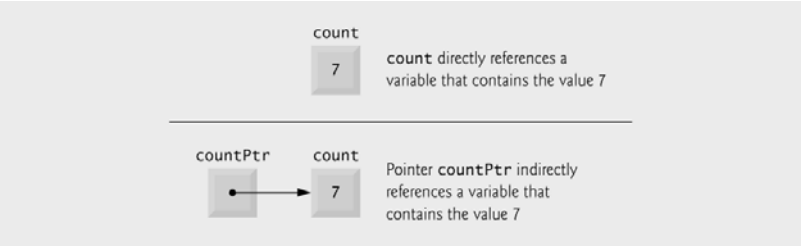


Fig. 7.1 | Directly and indirectly referencing a variable.

© 2007 Pearson Education, Inc. All rights reserved.

7.2 Pointer Variable Definitions and Initialization

- **Pointer definitions**
  - **`*` used with pointer variables**

```
int *myPtr;
```

  - **Defines a pointer to an `int` (pointer of type `int *`)**
  - **Multiple pointers require using a `*` before each variable definition**

```
int *myPtr1, *myPtr2;
```

  - **Can define pointers to any data type**
  - **Initialize pointers to `0`, `NULL`, or an address**
    - **`0` or `NULL` – points to nothing (`NULL` preferred)**

© 2007 Pearson Education, Inc. All rights reserved.

Common Programming Error 7.1

The asterisk (`*`) notation used to declare pointer variables does not distribute to all variable names in a declaration. Each pointer must be declared with the `*` prefixed to the name; e.g., if you wish to declare `xPtr` and `yPtr` as int pointers, use `int *xPtr, *yPtr;`.

© 2007 Pearson Education, Inc. All rights reserved.

Good Programming Practice 7.1

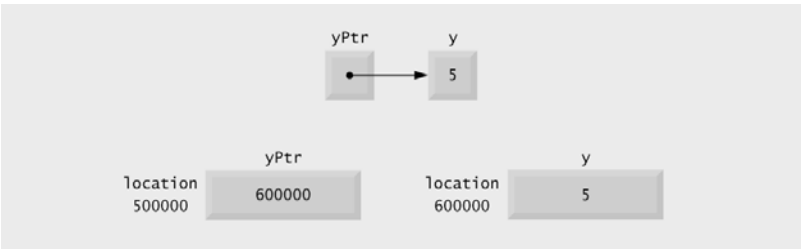
Include the letters `ptr` in pointer variable names to make it clear that these variables are pointers and thus need to be handled appropriately.

© 2007 Pearson Education, Inc. All rights reserved.

7.3 Pointer Operators

- **& (address operator)**
  - **Returns address of operand**

```
int y = 5;
int *yPtr;
yPtr = &y;     /* yPtr gets address of y */
yPtr "points to" y
```



© 2007 Pearson Education, Inc. All rights reserved.

7.3 Pointer Operators

- * (indirection/dereferencing operator)
  - Returns a synonym/alias of what its operand points to
  - `*yptr` returns `y` (because `yptr` points to `y`)
  - * can be used for assignment
    - Returns alias to an object

```
*yptr = 7;  /* changes y to 7 */
```

  - Dereferenced pointer (operand of *) must be an lvalue (no constants)
- * and & are inverses
  - They cancel each other out

© 2007 Pearson Education, Inc. All rights reserved.

Outline

fig07_04.c

```
3  #include <stdio.h>
4
5  int main( void )
6  {
7     int a;        /* a is an integer */
8     int *aPtr;    /* aPtr is a pointer to an integer */
9
10    a = 7;
11    aPtr = &a;    /* aPtr set to address of a */
12
13    printf( "The address of a is %p"
14            "\nThe value of aPtr is %p", &a, aPtr );
15
16    printf( "\n\nThe value of a is %d"
17            "\nThe value of *aPtr is %d", a, *aPtr );
18
19    printf( "\n\nShowing that * and & are complements of "
20            "each other\n&*aPtr = %p"
21            "\n*&aPtr = %p\n", &*aPtr, *&aPtr );
22
23    return 0; /* indicates successful termination */
24
25 } /* end main */
```

If **aPtr** points to **a**, then **&a** and **aPtr** have the same value.

a and ***aPtr** have the same value

&*aPtr and ***&aPtr** have the same value

```
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
```

© 2007 Pearson Education, Inc. All rights reserved.

Common Programming Error 7.2

Dereferencing a pointer that has not been properly initialized or that has not been assigned to point to a specific location in memory is an error. This could cause a fatal execution-time error, or it could accidentally modify important data and allow the program to run to completion with incorrect results.

© 2007 Pearson Education, Inc. All rights reserved.

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.

© 2007 Pearson Education, Inc. All rights reserved.

7.4 Calling Functions by Reference

- **Call by reference with pointer arguments**
  - **Pass address of argument using & operator**
  - **Allows you to change actual location in memory**
  - **Arrays are not passed with & because the array name is already a pointer**
- *** operator**
  - **Used as alias/nickname for variable inside of function**

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

  - `*number` **used as nickname for the variable passed**

© 2007 Pearson Education, Inc. All rights reserved.

Outline

fig07_06.c

```
1  /* Fig. 7.6: fig07_06.c
2     Cube a variable using call-by-value */
3  #include <stdio.h>
4
5  int cubeByValue( int n ); /* prototype */
6
7  int main( void )
8  {
9     int number = 5; /* initialize number */
10
11    printf( "The original value of number is %d", number );
12
13    /* pass number by value to cubeByValue */
14    number = cubeByValue( number );
15
16    printf( "\nThe new value of number is %d\n", number );
17
18    return 0; /* indicates successful termination */
19
20 } /* end main */
21
22 /* calculate and return cube of integer argument */
23 int cubeByValue( int n )
24 {
25    return n * n * n;   /* cube local variable n and return result */
26
27 } /* end function cubeByValue */
```

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

© 2007 Pearson Education, Inc. All rights reserved.

Outline

fig07_07.c

```
1  /* Fig. 7.7: fig07_07.c
2     Cube a variable using call-by-reference with a pointer argument */
3
4  #include <stdio.h>
5
6  void cubeByReference( int *nPtr ); /* prototype */
7
8  int main( void )
9  {
10    int number = 5; /* initialize number */
11
12    printf( "The original value of number is %d", number );
13
14    /* pass address of number to cubeByReference */
15    cubeByReference( &number );
16
17    printf( "\nThe new value of number is %d\n", number );
18
19    return 0; /* indicates successful termination */
20
21 } /* end main */
22
23 /* calculate cube of *nPtr; modifies variable number in main */
24 void cubeByReference( int *nPtr )
25 {
26    *nPtr = *nPtr * *nPtr * *nPtr;  /* cube *nPtr */
27 } /* end function cubeByReference */
```

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

Function prototype takes a pointer argument

Function **cubeByReference** is passed an address, which can be the value of a pointer variable

In this program, ***nPtr** is **number**, so this statement modifies the value of **number** itself.

© 2007 Pearson Education, Inc. All rights reserved.

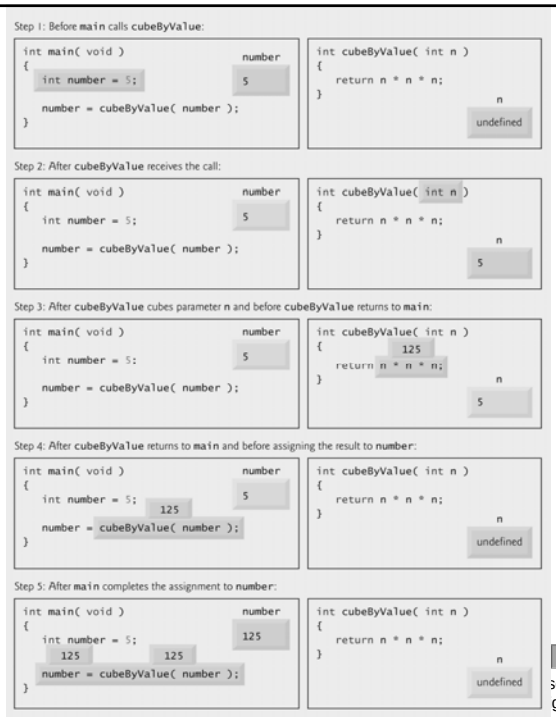


Fig. 7.8 | Analysis of a typical call-by-value.

son Education, ghts reserved.

Fig. 7.9 | Analysis of a typical call-by-reference with a pointer argument.

© 2007 Pearson Education, Inc. All rights reserved.

Common Programming Error 7.3

Not dereferencing a pointer when it is necessary to do so in order to obtain the value to which the pointer points is a syntax error.

© 2007 Pearson Education, Inc. All rights reserved.

Software Engineering Observation 7.2

Only one value can be altered in a calling function when call-by-value is used. That value must be assigned from the return value of the function. To modify multiple values in a calling function, call-by-reference must be used.

© 2007 Pearson Education, Inc. All rights reserved.

Common Programming Error 7.4

Being unaware that a function is expecting pointers as arguments for call-by-reference and passing arguments call-by-value. Some compilers take the values assuming they are pointers and dereference the values as pointers. At runtime, memory-access violations or segmentation faults are often generated. Other compilers catch the mismatch in types between arguments and parameters and generate error messages.

© 2007 Pearson Education, Inc. All rights reserved.

References and Reference Parameters

- **Reference Parameter**
  - **Another way to pass-by-reference (C++ only)**
  - **& placed after the parameter type in the function prototype and function header**
  - **Parameter name in the body of the called function actually refers to the original variable in the calling function (an alias)**
- **References can be used as aliases to other variables**
  - **Refer to same variable**

```
int count = 1; // declare integer variable count
Int &cRef = count; // create cRef as an alias for count
++cRef; // increment count (using its alias)
```

- **References must be initialized when declared**
  - **Otherwise, compiler error**
  - **Dangling reference**
    - **Reference to undefined variable**

© 2007 Pearson Education, Inc. All rights reserved.

Outline

fig03_21.cpp (1 of 1)

fig03_21.cpp output (1 of 1)

```
1  // Fig. 3.21: fig03_21.cpp
2  // References must be initialized.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int main()
9  {
10    int x = 3;
11
12    // y refers to (is an alias for) x
13    int &y = x;
14
15    cout << "x = " << x << endl << "y = " << y << endl;
16    y = 7;
17    cout << "x = " << x << endl << "y = " << y << endl;
18
19    return 0;  // indicates successful termination
20
21 } // end main
```

y declared as a reference to **x**.

```
x = 3
y = 3
x = 7
y = 7
```

© 2003 Prentice Hall, Inc.
All rights reserved.

Outline

fig03_22.cpp (1 of 1)

fig03_22.cpp output (1 of 1)

```
1   // Fig. 3.22: fig03_22.cpp
2   // References must be initialized.
3   #include <iostream>
4
5   using std::cout;
6   using std::endl;
7
8   int main()
9   {
10     int x = 3;
11     int &y;     // Error: y must be initialized
12
13     cout << "x = " << x << endl << "y = " << y << endl;
14     y = 7;
15     cout << "x = " << x << endl << "y = " << y << endl;
16
17     return 0;  // indicates successful termination
18
19  } // end main
```

Uninitialized reference – compiler error.

```
Borland C++ command-line compiler error message:
 Error E2304 Fig03_22.cpp 11: Reference variable 'y' must be
   initialized- in function main()

Microsoft Visual C++ compiler error message:
 D:\cpphtp4_examples\ch03\Fig03_22.cpp(11) : error C2530: 'y' :
   references must be initialized
```

© 2003 Prentice Hall, Inc.
All rights reserved.

Pointer v.s. Reference

```
void cubeByPtr(int *nPtr);

int main(void) {
  int number = 5;
  ......
  cubeByPtr(&number);
  ......
}

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

- **Different function prototype !!**
- **Dereference pointer in the calculation is not convenient**

```
void cubeByRef(int &nRef);

int main(void) {
  int number = 5;
  ......
  cubeByRef(number);
  ......
}

void cubeByRef(int &nRef) {
  nRef = nRef*nRef*nRef;
}
```

- **Different function prototype !!**
- **Use the variable in the same way**
- **Not easy to aware that it is a reference**

© 2007 Pearson Education, Inc. All rights reserved.

7.5 Using the `const` Qualifier with Pointers

- `const` qualifier
  - Variable cannot be changed
  - Use `const` if function does not need to change a variable
  - Attempting to change a `const` variable produces an error
- `const` pointers
  - Point to a constant memory location
  - Must be initialized when defined
  - `int *const myPtr = &x;`
    - Type `int *const` – constant pointer to an `int`
  - `const int *myPtr = &x;`
    - Modifiable pointer to a `const int`
  - `const int *const Ptr = &x;`
    - `const` pointer to a `const int`
    - `x` itself can be changed, but not `*Ptr`

© 2007 Pearson Education, Inc. All rights reserved.

Outline

fig07_10.c

```
5  #include <stdio.h>
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
18    return 0; /* indicates successful termination */
19
20 } /* end main */
21
22 /* convert string to uppercase letters */
23 void convertToUppercase( char *sPtr )
24 {
25    while ( *sPtr != '\0' ) { /* current character is not '\0' */
26
27       if ( islower( *sPtr ) ) {    /* if character is lowercase, */
28          *sPtr = toupper( *sPtr ); /* convert to uppercase */
29       } /* end if */
30
31       ++sPtr;  /* move sPtr to the next character */
32    } /* end while */
33
34 } /* end function convertToUppercase */
```

Both **sPtr** and ***sPtr** are modifiable

Both **sPtr** and ***sPtr** are modified by the **convertToUppercase** function

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

© 2007 Pearson Education, Inc. All rights reserved.

Outline

fig07_11.c

```
4
5  #include <stdio.h>
6
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
18    return 0; /* indicates successful termination */
19
20 } /* end main */
21
22 /* sPtr cannot modify the character to which it points,
23    i.e., sPtr is a "read-only" pointer */
24 void printCharacters( const char *sPtr )
25 {
26    /* loop through entire string */
27    for ( ; *sPtr != '\0'; sPtr++ ) { /* no initialization */
28       printf( "%c", *sPtr );
29    } /* end for */
30
31 } /* end function printCharacters */
```

Pointer variable **sPtr** is modifiable, but the data to which it points, ***sPtr**, is not

sPtr is modified by function **printCharacters**

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

© 2007 Pearson Education, Inc. All rights reserved.

Outline

fig07_12.c

```
1  /* Fig. 7.12: fig07_12.c
2     Attempting to modify data through a
3     non-constant pointer to constant data. */
4  #include <stdio.h>
5  void f( const int *xPtr ); /* prototype */
6
7
8  int main( void )
9  {
10    int y;       /* define y */
11
12    f( &y );     /* f attempts illegal modification */
13
14    return 0;    /* indicates successful termination */
15
16 } /* end main */
17
18 /* xPtr cannot be used to modify the
19    value of the variable to which it points */
20 void f( const int *xPtr )
21 {
22    *xPtr = 100;  /* error: cannot modify a const object */
23 } /* end function f */
```

Pointer variable **xPtr** is modifiable, but the data to which it points, ***xPtr**, is not

***xPtr** has the **const** qualifier, so attempting to modify its value causes an error

```
Compiling...
FIG07_12.c
c:\books\2006\chtp5\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)
```

© 2007 Pearson Education, Inc. All rights reserved.

Outline

fig07_13.c

```
1  /* Fig. 7.13: fig07_13.c
2     Attempting to modify a constant pointer to non-constant data */
3  #include <stdio.h>
4
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
17    return 0; /* indicates successful termination */
18
19 } /* end main */
```

Pointer **ptr** is not modifiable, but the data to which it points, ***ptr**, can be changed

```
Compiling...
FIG07_13.c
c:\books\2006\chtp5\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)
```

© 2007 Pearson Education, Inc. All rights reserved.

Outline

fig07_14.c

```
1  /* Fig. 7.14: fig07_14.c
2     Attempting to modify a constant pointer to constant data. */
3  #include <stdio.h>
4
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
20    return 0; /* indicates successful termination */
21
22 } /* end main */
```

Neither pointer **sPtr** nor the data to which it points, ***sPtr**, is modifiable

```
Compiling...
FIG07_14.c
c:\books\2006\chtp5\Examples\ch07\FIG07_14.c(17) : error C2166: l-value
   specifies const object
c:\books\2006\chtp5\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)
```

© 2007 Pearson Education, Inc. All rights reserved.

Error-Prevention Tips

7.2: Use call-by-value to pass arguments to a function unless the caller explicitly requires the called function to modify the value of the argument variable in the caller's environment. This prevents accidental modification of the caller's arguments.

7.3: If a variable does not (or should not) change in the body of a function to which it is passed, its pointer should be declared const to ensure that it is not accidentally modified.

© 2007 Pearson Education, Inc. All rights reserved.

Performance Tip 7.1

Pass large objects such as structures using pointers to constant data to obtain the performance benefits of call-by-reference and the security of call-by-value.

© 2007 Pearson Education, Inc. All rights reserved.

7.6 Bubble Sort Using Call-by-Reference

- **Implement bubble sort using pointers**
  - **Swap two elements**
  - **`swap` function must receive address (using `&`) of array elements**
    - **Array elements have call-by-value default**
  - **Using pointers and the `*` operator, `swap` can switch array elements**
- **Psuedocode**

  Initialize array

  print data in original order

  Call function bubble sort

  print sorted array

  Define bubble sort

© 2007 Pearson Education, Inc. All rights reserved.

Outline

fig07_15.c

(1 of 3)

```
1  /* Fig. 7.15: fig07_15.c
2     This program puts values into an array, sorts the values into
3     ascending order, and prints the resulting array. */
4  #include <stdio.h>
5  #define SIZE 10
6
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 */
```

© 2007 Pearson Education, Inc. All rights reserved.

```
31
32    printf( "\n" );
33
34    return 0; /* indicates successful termination */
35
36 } /* end main */
37
38 /* sort an array of integers using bubble sort algorithm */
39 void bubbleSort( int * const array, const int size )
40 {
41    void swap( int *element1Ptr, int *element2Ptr ); /* prototype */
42    int pass; /* pass counter */
43    int j;    /* comparison counter */
44
45    /* loop to control passes */
46    for ( pass = 0; pass < size - 1; pass++ ) {
47
48       /* loop to control comparisons during each pass */
49       for ( j = 0; j < size - 1; j++ ) {
50
51          /* swap adjacent elements if they are out of order */
52          if ( array[ j ] > array[ j + 1 ] ) {
53             swap( &array[ j ], &array[ j + 1 ] );
54          } /* end if */
55
56       } /* end inner for */
57
58    } /* end outer for */
59
60 } /* end function bubbleSort */
```

Outline

fig07_15.c

(2 of 3)

© 2007 Pearson Education, Inc. All rights reserved.


```
61
62 /* swap values at memory locations to which element1Ptr and
63    element2Ptr point */
64 void swap( int *element1Ptr, int *element2Ptr )
65 {
66    int hold = *element1Ptr;
67    *element1Ptr = *element2Ptr;
68    *element2Ptr = hold;
69 } /* end function swap */
```

Function **swap** changes the values of the **int**s that the two pointers point to

```
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

(3 of 3)

© 2007 Pearson Education, Inc. All rights reserved.

35

Software Engineering Observation 7.3

Placing function prototypes in the definitions of other functions enforces the principle of least privilege by restricting proper function calls to the functions in which the prototypes appear.

© 2007 Pearson Education, Inc. All rights reserved.

36

Software Engineering Observation 7.4

When passing an array to a function, also pass the size of the array. This helps make the function reusable in many programs.

© 2007 Pearson Education, Inc. All rights reserved.