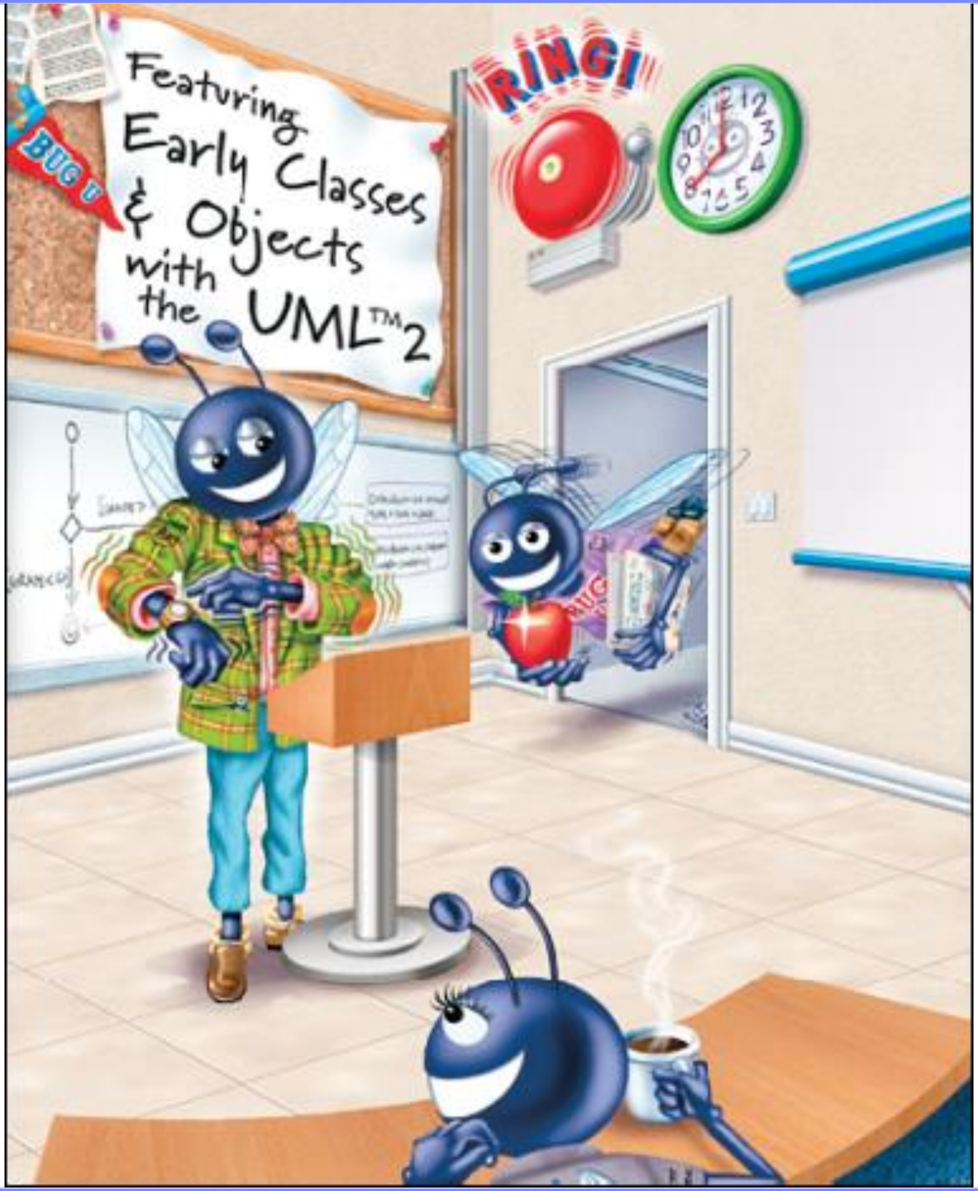
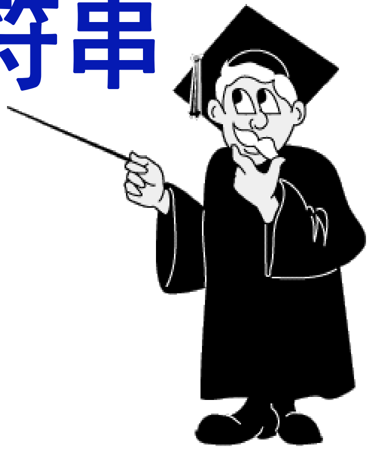


Lecture 5:
指针

第五讲 指针和基于指针的字符串



学习目标：

- 指针和引用的异同
- 指针作为参数传递给函数
- 基于指针的 C 风格的字符串
- 指针和数组的关系
- 函数指针

1 Introduction

- 指针(Pointers)
  - 功能强大但难于掌握
  - 可以用来执行 pass-by-reference
  - 可以用来创建和操纵动态数据结构
  - 与数组和字符串有着密切的关系

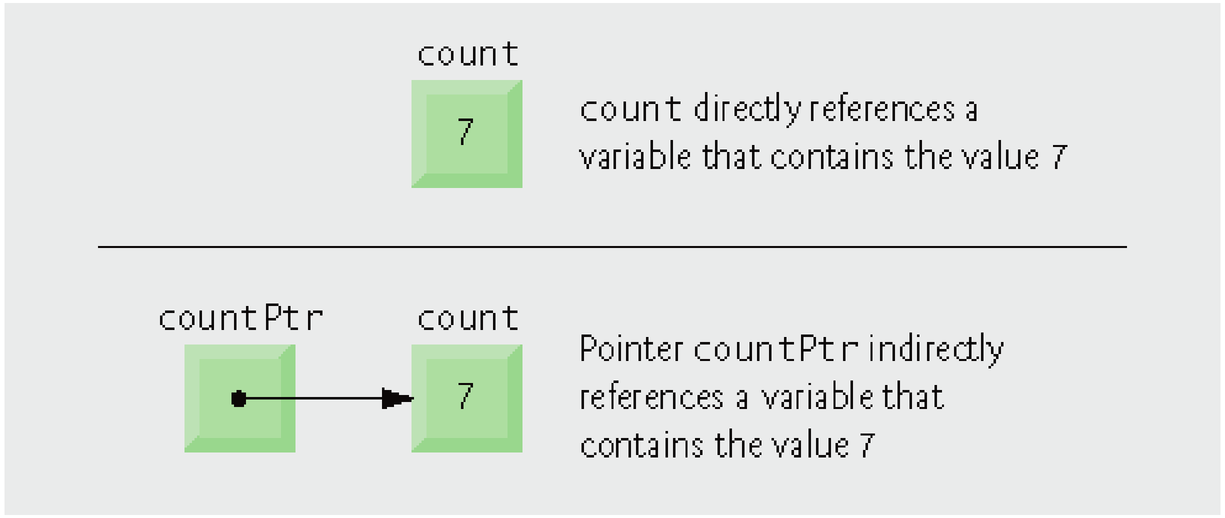
2 Pointer Variable Declarations and Initialization

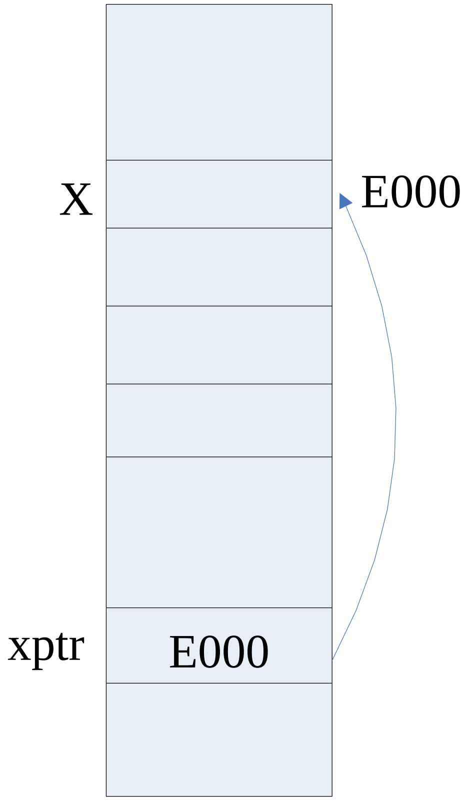
- Pointer variables
  - 将内存地址作为变量值
    - ✓ 通常变量包含特定的值(直接引用)
    - ✓ 指针包含变量的地址值(间接引用)
- Indirection
  - 通过指针来引用变量的值

2 Pointer Variable Declarations and Initialization

- Pointer declarations
  - * 指示一个变量为指针变量
    - ✓ 声明一个int 型变量的指针：int *myPtr;
    - ✓ 声明多个指针变量：int *myPtr1, *myPtr2;
- Pointer initialization
  - 初始化为 0, NULL 或一个地址
  - 指针初始化的目的是防止它指向内存中非用户数据区，从而引起一系列的错误。

2 Pointer Variable Declarations and Initialization


count
7
count directly references a variable that contains the value 7
countPtr
count
7
Pointer countPtr indirectly references a variable that contains the value 7



2 Pointer Variable Declarations and Initialization

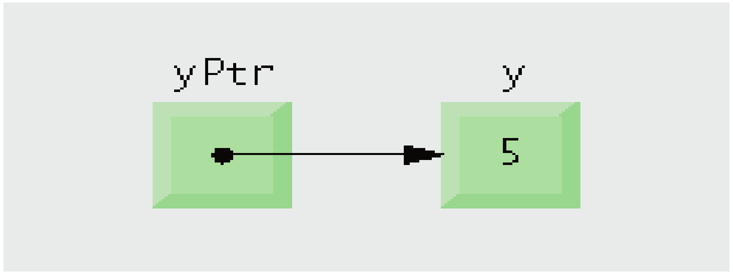
常见编程错误：声明指针变量时，没有在指针变量前面加上“*”。

良好编程习惯：在指针变量名中包含字母 Ptr能够更清楚地表示这个变量是指针变量。

错误预防技巧：指针初始化是为了防止指向未知的、未经初始化的甚至是系统使用的内存区域。

3 Pointer Operators

- **取地址运算符 (&)**
  - 返回操作数的地址
  - int y = 5;
    int *yPtr;
    yPtr = &y;   //变量 yPtr “指向” y

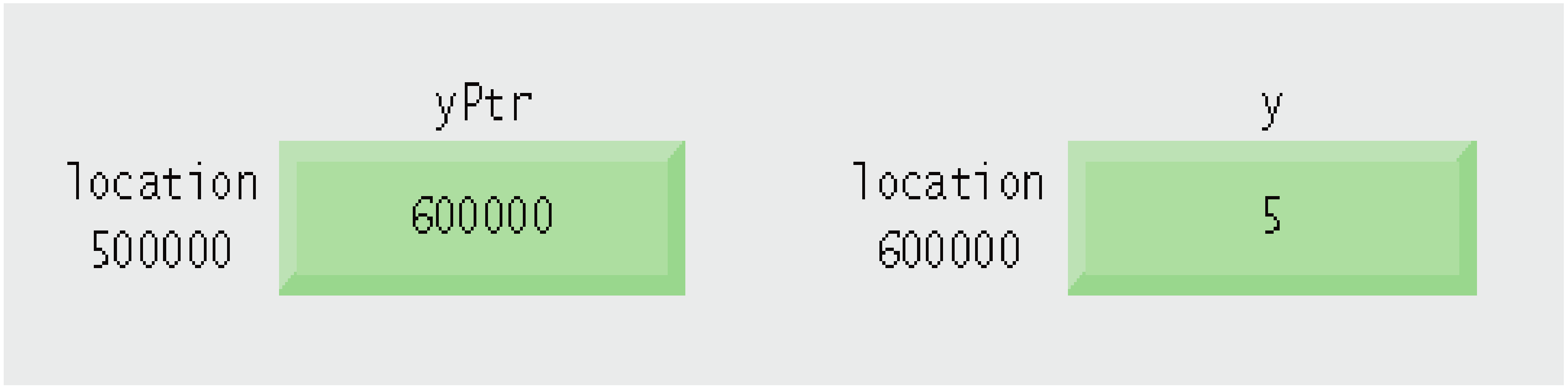


3 Pointer Operators

- **取值运算符(*)**
  - 也称为 indirection operator or dereferencing operator
(间接运算符、间接引用运算符)
  - *yPtr 返回 y (取指针yptr所指向地址的值)
  - Dereferenced pointer is an *lvalue*

    *yPtr = 5;
- *** and & are inverses of each other(互逆操作)**

3 Pointer Operators



```cpp
// Fig. 8.4: fig08_04.cpp
// Using the & and * operators.
#include <iostream>
using std::cout;
using std::endl;

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

   a = 7; // assigned 7 to a
   aPtr = &a; // assign the address of a to aPtr
```

Variable **aPtr** is a point to an **int**

Initialize **aPtr** with the address of variable **a**

```
int a=7;

int *aptr=a;
```

```
int a;

int *aptr=a;

a=7;
```

```
   cout << "The address of a is " << &a
      << "\nThe value of aPtr is " << aPtr;
   cout << "\n\nThe value of a is " << a
      << "\nThe value of *aPtr is " << *aPtr;
   cout << "\n\nShowing that * and & are inverses of "
      << "each other.\n&*aPtr = " << &*aPtr
      << "\n*&aPtr = " << *&aPtr << endl;
   return 0; // indicates successful termination
} // end main
```

Address of **a** and the value of **aPtr** are identical

Value of **a** and the dereferenced **aPtr** are identical

* and **&** are inverses of each other

```
The address of a is 0012F580
The value of aPtr is 0012F580

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

Showing that * and & are inverses of each other.
&*aPtr = 0012F580
*&aPtr = 0012F580
```

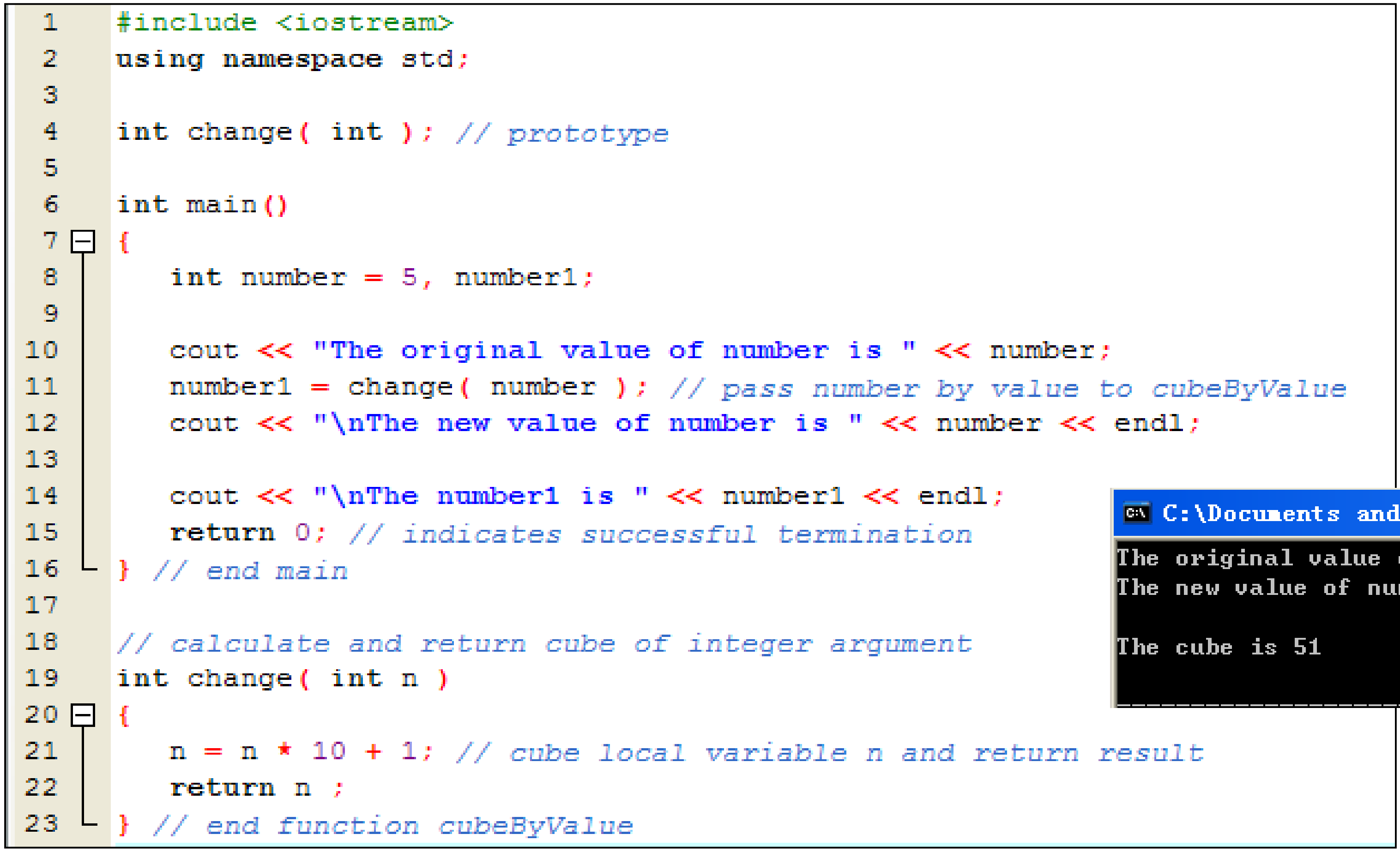
* and **&** are inverses; same result when both are applied to **aPtr**

4 Passing Arguments to Functions by Reference with Pointers

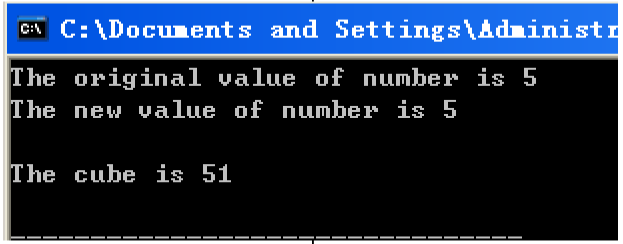
- 三种向函数传递参数的方式
  - Pass-by-value(值传递)
  - Pass-by-reference with reference arguments(以引用为参数的引用传递)
  - Pass-by-reference with pointer arguments(以指针为参数的引用传递)

值传递方式

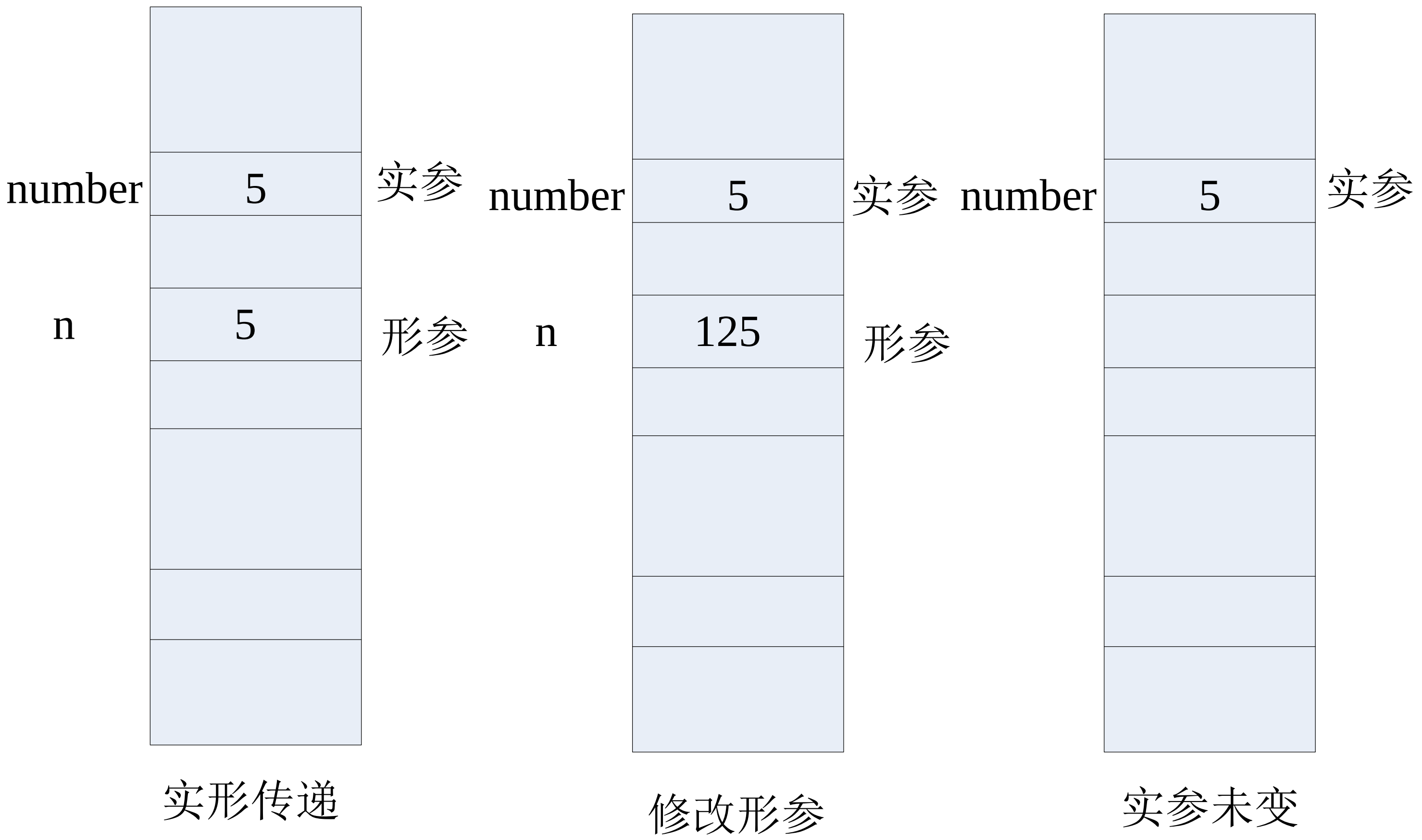
- 优点：保护实参，不受被调用函数的影响
- 缺点：被调用函数无法修改实参



```
#include <iostream>
using namespace std;

int change( int ); // prototype

int main()
{
   int number = 5, number1;

   cout << "The original value of number is " << number;
   number1 = change( number ); // pass number by value to cubeByValue
   cout << "\nThe new value of number is " << number << endl;

   cout << "\nThe number1 is " << number1 << endl;
   return 0; // indicates successful termination
} // end main

// calculate and return cube of integer argument
int change( int n )
{
   n = n * 10 + 1; // cube local variable n and return result
   return n ;
} // end function cubeByValue
```



```
C:\Documents and Settings\Administr
The original value of number is 5
The new value of number is 5

The cube is 51
```


number
5
实参
n
5
形参
实形传递
number
5
实参
n
125
形参
修改形参
number
5
实参
实参未变

```cpp
// Fig. 8.6: fig08_06.cpp
// Cube a variable using pass-by-value.
#include <iostream>
using std::cout;
using std::endl;

int cubeByValue( int ); // prototype

int main()
{
   int number = 5;

   cout << "The original value of number is " << number;

   number = cubeByValue( number ); // pass number by value to cubeByValue
   cout << "\nThe new value of number is " << number << endl;
   return 0; // indicates successful termination
} // end main

// calculate and return cube of integer ar
int cubeByValue( int n )
{
   return n * n * n; // cube local variable n and return result
} // end function cubeByValue
```

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

Pass number by value; result returned by **cubeByValue**

cubeByValue receives parameter passed-by-value

Cubes local variable **n** and **return** the result

```cpp
// Fig. 8.7: fig08_07.cpp
// Cube a variable using pass-by-reference with a pointer argument.
#include <iostream>
using std::cout;
using std::endl;

void cubeByReference( int * ); // prototype

int main()
{
   int number = 5;

   cout << "The original value of number is " << number;

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

   cout << "\nThe new value of number is " << number << endl;
   return 0; // indicates successful termination
} // end main

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

Prototype indicates parameter is a pointer to an **int**

Apply address operator **&** to pass address of **number** to **cubeByReference**

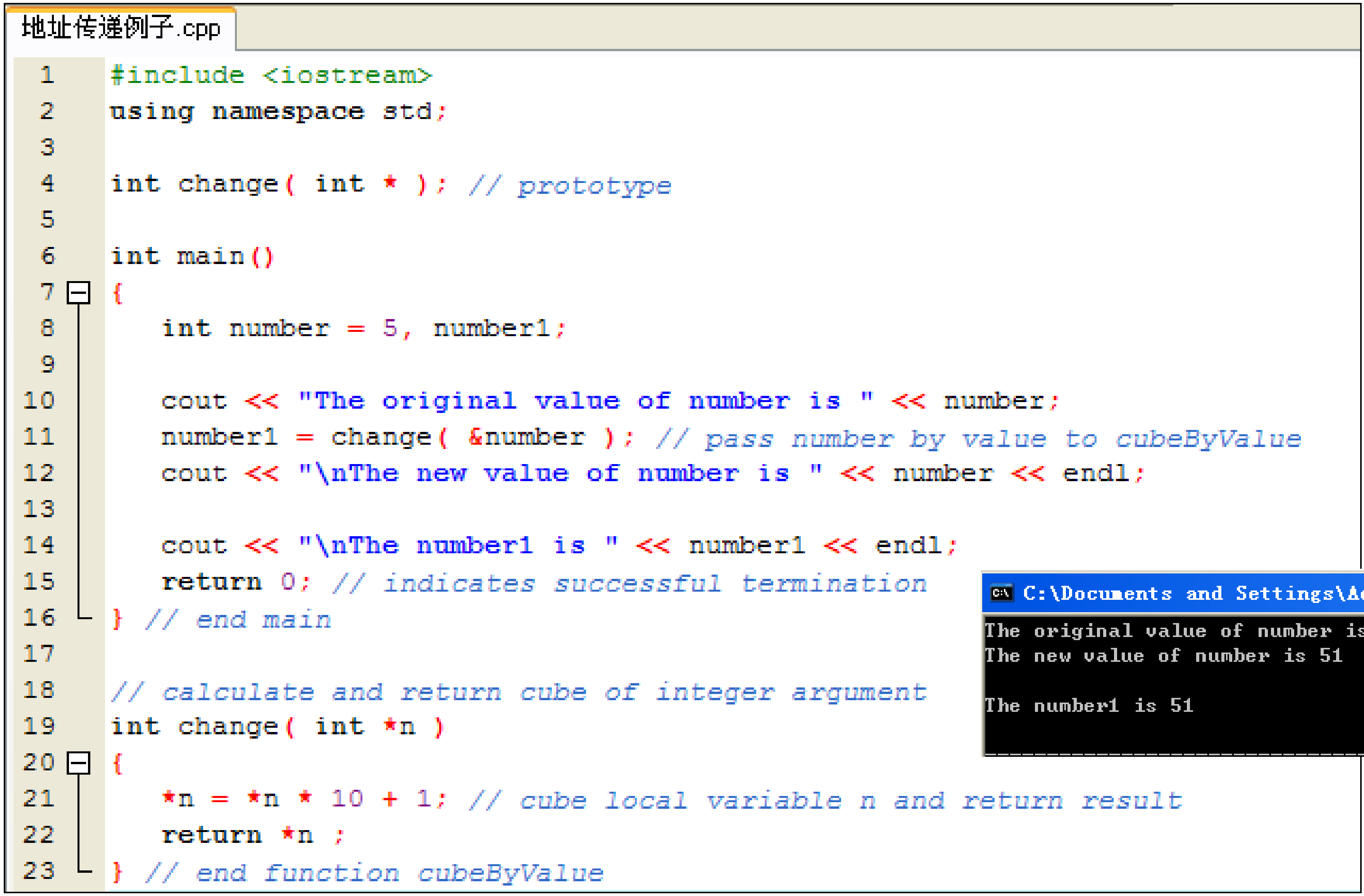
cubeByReference modifies variable **number**

cubeByReference receives address of an **int** variable, i.e., a pointer to an **int**

Modify and access **int** variable using indirection operator *

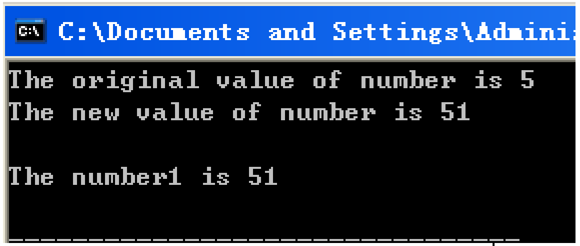
```
The original value of number is 5
The new value of number is 125
```

C++ How to Program



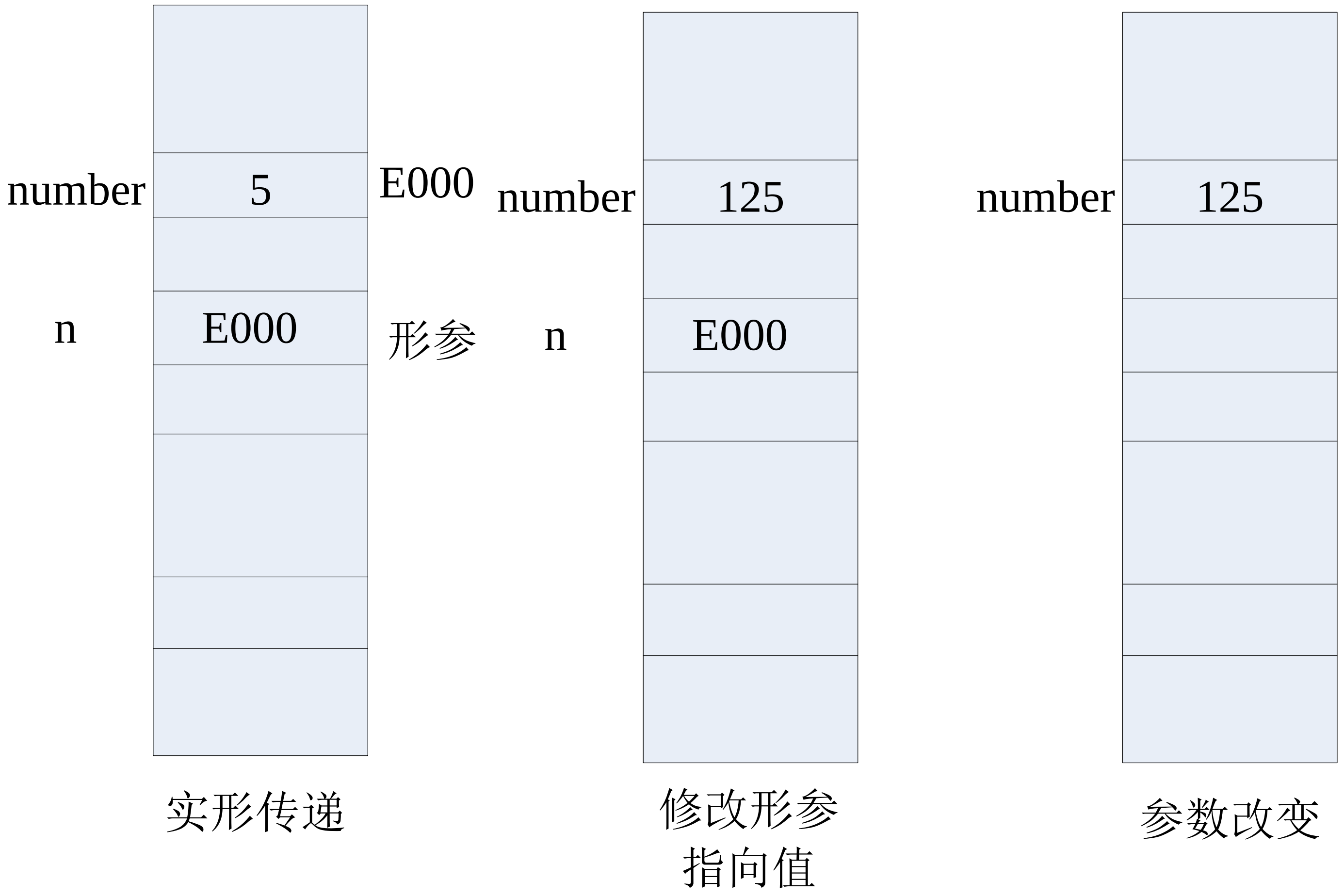
地址传递例子.cpp

```cpp
#include <iostream>
using namespace std;

int change( int * ); // prototype

int main()
{
    int number = 5, number1;

    cout << "The original value of number is " << number;
    number1 = change( &number ); // pass number by value to cubeByValue
    cout << "\nThe new value of number is " << number << endl;

    cout << "\nThe number1 is " << number1 << endl;
    return 0; // indicates successful termination
} // end main

// calculate and return cube of integer argument
int change( int *n )
{
    *n = *n * 10 + 1; // cube local variable n and return result
    return *n ;
} // end function cubeByValue
```



C:\Documents and Settings\Admini

```
The original value of number is 5
The new value of number is 51

The number1 is 51
```

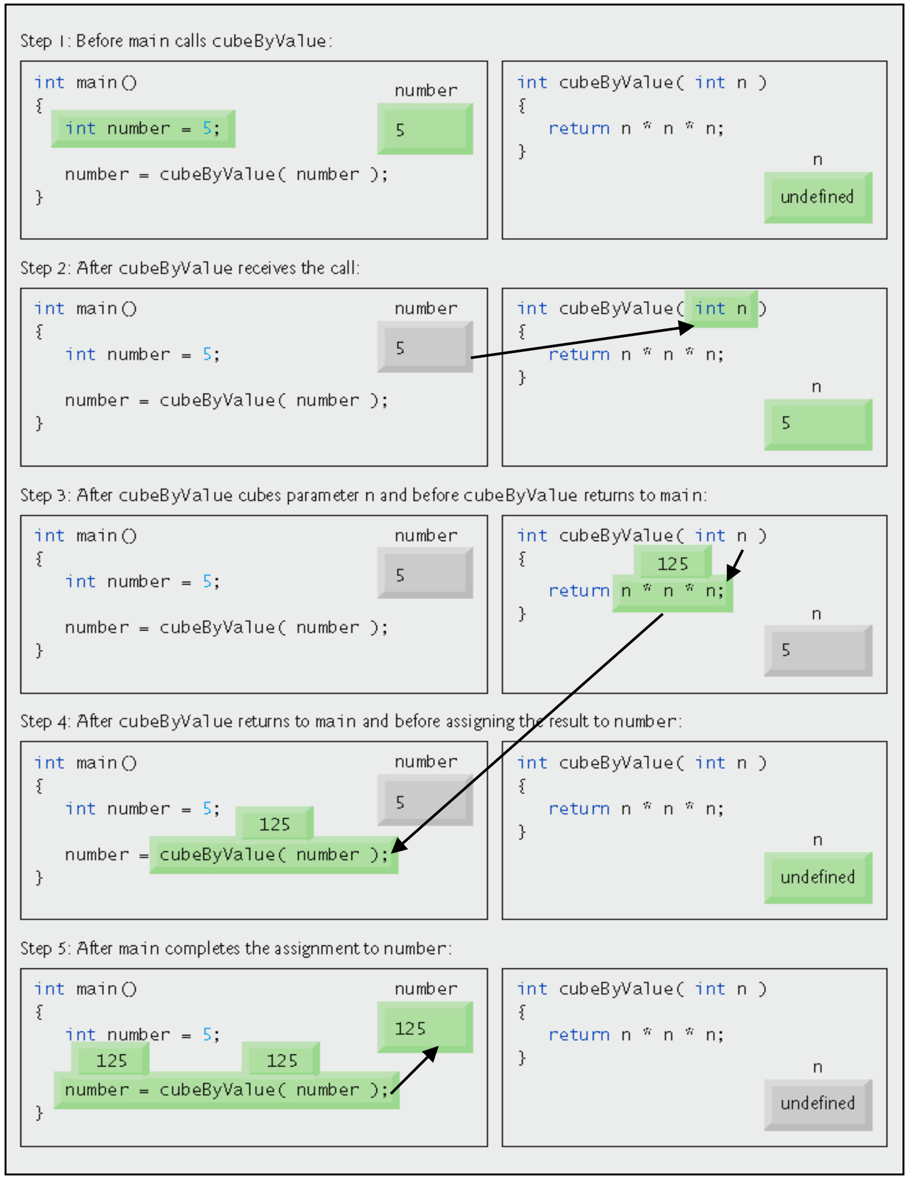

number
5
E000
n
E000
形参
实形传递
number
125
n
E000
修改形参
指向值
number
125
参数改变

4 Passing Arguments to Functions by Reference with Pointers

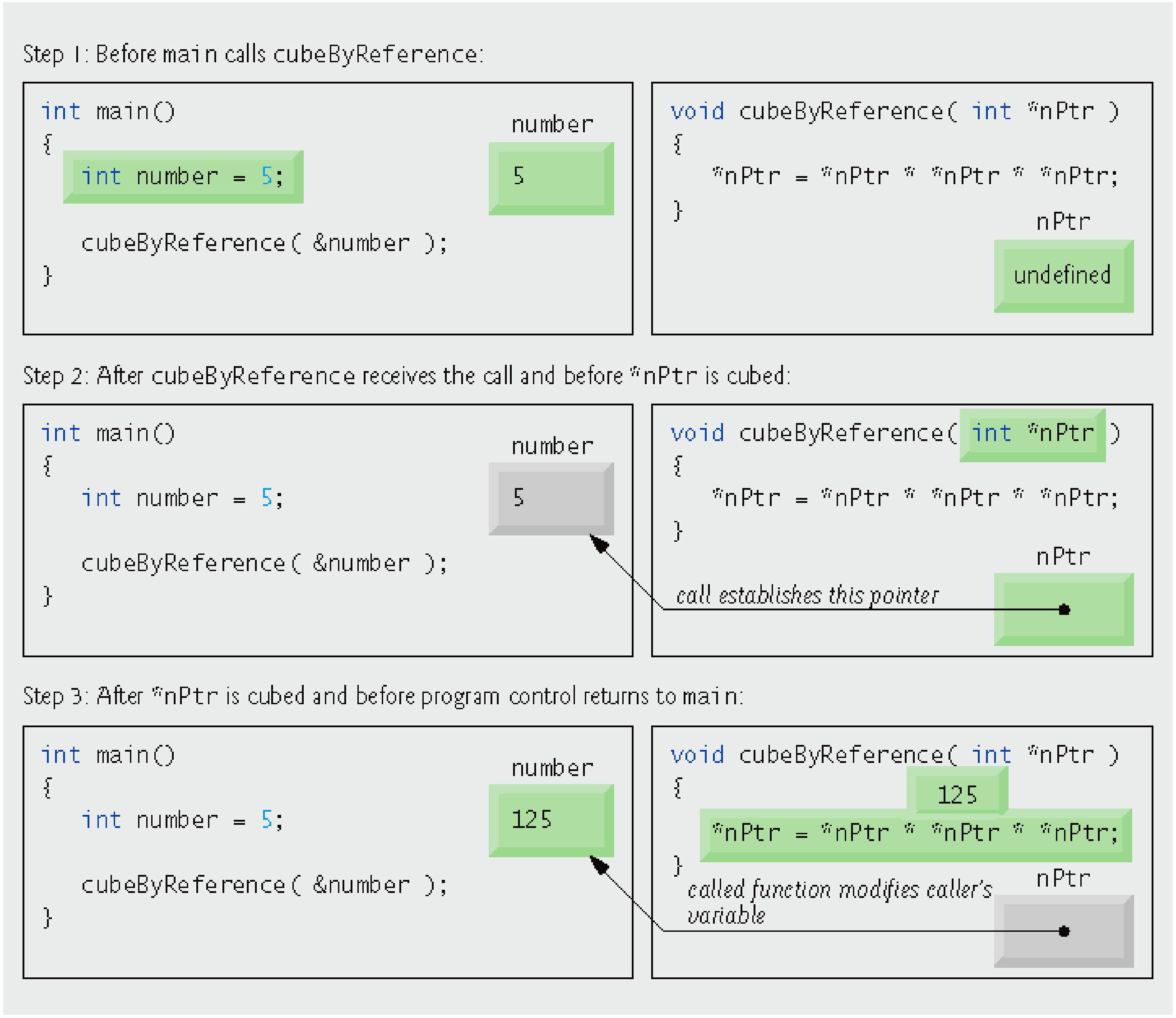
- 一个函数只能返回一个值
- 使用引用参数向函数进行参数传递
  - 函数可以修改参数的原始值
  - 函数可以返回“多个值”

4 Passing Arguments to Functions by Reference with Pointers

- **使用指针参数按引用传递**
  - 使用 & 运算符传递参数地址
  - 如果实参是数组，因为数组名即为数组首地址，故数组名前无须再加“&”
  - * 运算符在函数内部用做参数的别名来使用


Step 1: Before main calls cubeByValue:
int main()
{
int number = 5;
number = cubeByValue(number);
}
number
5
int cubeByValue(int n)
{
return n * n * n;
}
n
undefined
Step 2: After cubeByValue receives the call:
number
5
n
5
Step 3: After cubeByValue cubes parameter n and before cubeByValue returns to main:
125
Step 4: After cubeByValue returns to main and before assigning the result to number:
125
undefined
Step 5: After main completes the assignment to number:
125
125
125
undefined

Pass-by-value analysis of the program


Step 1: Before main calls cubeByReference:
int main()
{
int number = 5;
cubeByReference(&number);
}
number
5
void cubeByReference(int *nPtr)
{
*nPtr = *nPtr * *nPtr * *nPtr;
}
nPtr
undefined
Step 2: After cubeByReference receives the call and before *nPtr is cubed:
int main()
{
int number = 5;
cubeByReference(&number);
}
number
5
void cubeByReference(int *nPtr)
{
*nPtr = *nPtr * *nPtr * *nPtr;
}
nPtr
call establishes this pointer
Step 3: After *nPtr is cubed and before program control returns to main:
int main()
{
int number = 5;
cubeByReference(&number);
}
number
125
void cubeByReference(int *nPtr)
{
125
*nPtr = *nPtr * *nPtr * *nPtr;
}
called function modifies caller's variable
nPtr

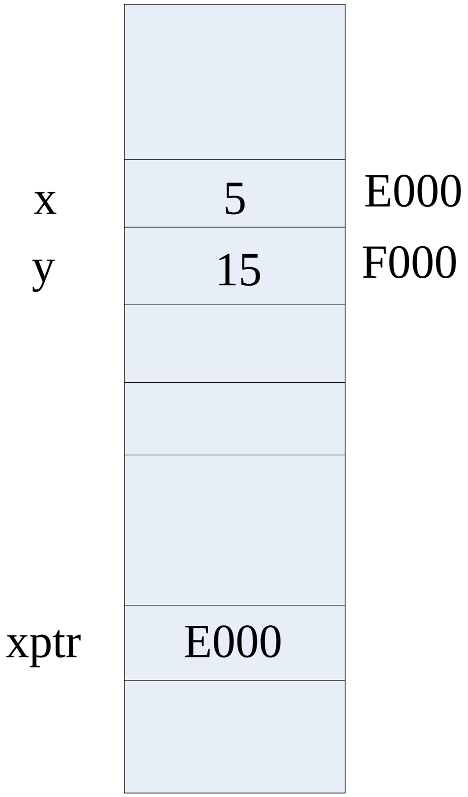
Pass-by-reference analysis (with a pointer argument) of the program

5 Using const with Pointers

- Principle of least privilege (最低权限原则)
  - 只需授予函数完成任务所需要的权限即可
  - 例如：打印数组元素的函数
    - ✓ 数组元素应为 const，不需要修改的权限
    - ✓ 数组长度应为 const，不需要修改的权限

5 Using const with Pointers

- const pointers(写在哪个前面就是修饰哪个)
  - Constant pointer to a non-constant int
    - int **const myPtr* = &x;
  - Non-constant pointer to a constant int
    - *const int* *myPtr = &x;
  - Constant pointer to a constant int
    - const int *const Ptr = &x;

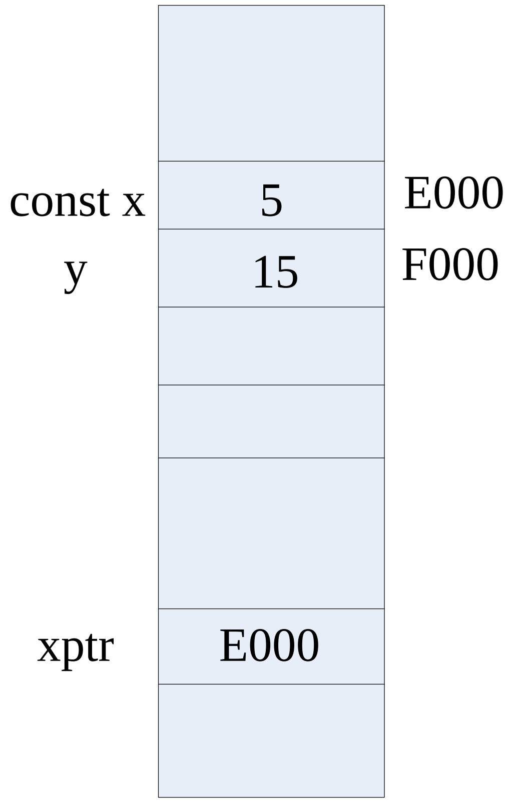


1. 非 const int , 非 const pointer

x=5 ; // 合法修改

*xptr = 20 ; // 合法修改

xptr = &y ; // 合法修改

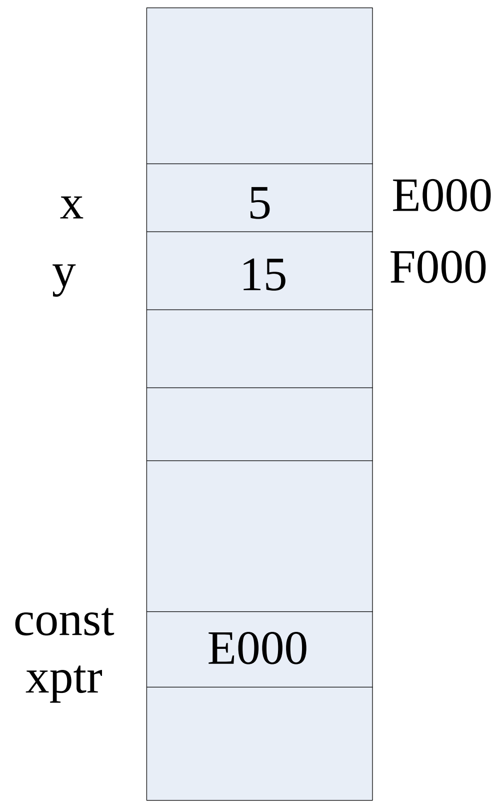


2. const int , 非 const pointer

x=15 ;   // 非法修改

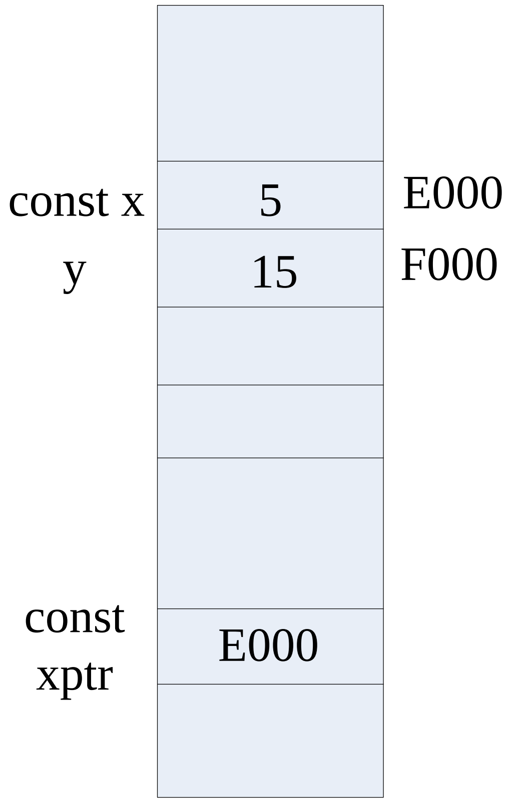
*xptr = 20 ;   // 非法修改

xptr = &y ;   // 合法修改



3. 非const int , const pointer

```
x=5 ;              // 合法修改
*xptr = 20 ;       //  合法修改
xptr = &y ;        //  非法修改
```



4. const int , const pointer

```
x=5 ;             // 非法修改
*xptr = 20 ;      //  非法修改
xptr = &y ;       //  非法修改
```

```cpp
// Fig. 8.10: fig08_10.cpp
// Converting lowercase letters to uppercase letters
// using a non-constant pointer to non-constant data.
#include <iostream>
using std::cout;
using std::endl;

#include <cctype> // prototypes for islower and toupper
using std::islower;
using std::toupper;

void convertToUppercase( char * );

int main()
{
   char phrase[] = "characters and $32.98";

   cout << "The phrase before conversion is: " << phrase;
   convertToUppercase( phrase );
   cout << "\nThe phrase after conversion is:  " << phrase << endl;
   return 0; // indicates successful termination
} // end main
```

非constant 数据，
非 constant 指针

convertToUppercase modifies variable **phrase**

情况1：非静态指针传送非静态参数。

```

// convert string to uppercase letters
void convertToUppercase( char *sPtr )
{
   while ( *sPtr != '\0' ) // loop while current character is not '\0'
   {
      if ( islower( *sPtr ) ) // if character
         *sPtr = toupper( *sPtr ); // convert to uppercase

      sPtr++; // move sPtr to next character in string
   } // end while
} // end function convertToUppercase
```

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

Parameter **sPtr** is a nonconstant pointer to nonconstant data

Function **islower** returns **true** if the character is lowercase

Function **toupper** returns corresponding uppercase character if original character is lowercase; otherwise **toupper** returns the original character

Modify the memory address stored in **sPtr** to point to the next element of the array

```cpp
// Fig. 8.11: fig08_11.cpp
// Printing a string one character at a time using
// a non-constant pointer to constant data.
#include <iostream>
using std::cout;
using std::endl;

void printCharacters( const char * ); // print using pointer to const data

int main()
{
   const char phrase[] = "print characters of a string";

   cout << "The string is:\n";
   printCharacters( phrase ); // print characters in phrase
   cout << endl;
   return 0; // indicates successful termination
} // end main

// sPtr can be modified, but it cannot modify the character to which
// it points, i.e., sPtr is a "read-only" pointer
void printCharacters( const char *sPtr )
{
   for ( ; *sPtr != '\0'; sPtr++ ) // no initialization
      cout << *sPtr; // display character without modification
} // end function printCharacters
```

constant 数据，
非 constant 指针

Pass pointer **phrase** to function **printCharacters**

sPtr is a nonconstant pointer to constant data; it cannot modify the character to which it points

Increment **sPtr** to point to the next character

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

情况2：非静态指针传送静态参数。

```
// Fig. 8.12: fig08_12.cpp
// Attempting to modify data through a
// non-constant pointer to constant data.

void f( const int * ); // prototype

int main()
{
   int y;

   f( &y ); // f attempts illegal modification
   return 0; // indicates successful termination
} // end main
```

Parameter is a nonconstant pointer to constant data

Pass the address of **int** variable **y** to attempt an illegal modification

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

Attempt to modify a **const** object pointed to by **xPtr**

Borland C++ command-line compiler error message:

```
Error E2024 fig08_12.cpp 18:
   Cannot modify a const object in function f(const int *)
```

Microsoft Visual C++ compiler error message:

```
c:\cpphtp5_examples\ch08\Fig08_12\fig08_12.cpp(18) :
   error C2166: l-value specifies const object
```

GNU C++ compiler error message:

```
fig08_12.cpp: In function `void f(const int*)':
fig08_12.cpp:18: error: assignment of read-only location
```

Error produced when attempting to compile

静态参数在被调函数内被修改----非法。

```
 1  // Fig. 8.13: fig08_13.cpp
 2  // Attempting to modify a constant pointer to non-constant data.
 3
 4  int main()
 5  {
 6     int x, y;
 7
 8     // ptr is a constant pointer to an integer that can
 9     // be modified through ptr, but ptr always points to the
10     // same memory location.
11     int * const ptr = &x; // const pointer must b
12
13     *ptr = 7; // allowed: *ptr is not const
14     ptr = &y; // error: ptr is const; cannot assign to it a new address
15     return 0; // indicates successful termination
16  } // end main
```

非constant 数据，
constant 指针

可以通过指针修改非const数据

但不能修改const指针的取值。
即不能指向另一个变量

Borland C++ command-line compiler error message:

```
Error E2024 fig08_13.cpp 14: Cannot modify a const object in function main()s
```

Microsoft Visual C++ compiler error message:

```
c:\cpphtp5e_examples\ch08\Fig08_13\fig08_13.cpp(14) : error C2166:
   l-value specifies const object
```

GNU C++ compiler error message:

```
fig08_13.cpp: In function `int main()':
fig08_13.cpp:14: error: assignment of read-only variable `ptr'
```

Line 14 generates a compiler error by attempting to assign a new address to a constant pointer

```
// Fig. 8.14: fig08_14.cpp
// Attempting to modify a constant pointer to constant data.
#include <iostream>
using std::cout;
using std::endl;

int main()
{
   int x = 5, y;

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

   cout << *ptr << endl;

   *ptr = 7; // error: *ptr is const; cannot assign new value
   ptr = &y; // error: ptr is const; cannot assign new address
   return 0; // indicates successful termination
} // end main
```

constant 数据，
constant 指针

不能修改 const 数据

不能修改 const 指针，即不能指向另一个变量

Borland C++ command-line compiler error message:

```
Error E2024 fig08_14.cpp 18: Cannot modify a const object in function main()
Error E2024 fig08_14.cpp 19: Cannot modify a const object in function main()
```

Microsoft Visual C++ compiler error message:

```
c:\cpphtp5e_examples\ch08\Fig08_14\fig08_14.cpp(18) : error C2166:
   l-value specifies const object
c:\cpphtp5e_examples\ch08\Fig08_14\fig08_14.cpp(19) : error C2166:
   l-value specifies const object
```

GNU C++ compiler error message:

```
fig08_14.cpp: In function `int main()':
fig08_14.cpp:18: error: assignment of read-only location
fig08_14.cpp:19: error: assignment of read-only variable `ptr'
```

Line 18 generates a compiler error by attempting to modify a constant object

Line 19 generates a compiler error by attempting to assign a new address to a constant pointer

6 Selection Sort Using Pass-by-Reference

- Implement `selectionSort` using pointers
  - Selection sort algorithm
    - Swap smallest element with the first element
    - Swap second-smallest element with the second element
    - Etc.
  - Want function `swap` to access array elements
    - Individual array elements: scalars
      - Passed by value by default
    - Pass by reference via pointers using address operator &

```
// Fig. 8.15: fig08_15.cpp
// This program puts values into an array, sorts the values into
// ascending order and prints the resulting array.
#include <iostream>
using std::cout;
using std::endl;

#include <iomanip>
using std::setw;

void selectionSort( int * const, const int ); // prototype,前面数组名，后面数组大小
void swap( int * const, int * const ); // prototype

int main()
{
   const int arraySize = 10;
   int a[ arraySize ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };

   cout << "Data items in original order\n";

   for ( int i = 0; i < arraySize; i++ )
      cout << setw( 4 ) << a[ i ];    // 排序之前显示

   selectionSort( a, arraySize ); // sort the array

   cout << "\nData items in ascending order\n";

   for ( int j = 0; j < arraySize; j++ )
      cout << setw( 4 ) << a[ j ];    // 排序之后显示
```

```cpp

   cout << endl;
   return 0; // indicates successful termination
} // end main

// function to sort an array
void selectionSort( int * const array, const int size )
{
   int smallest; // index of smallest element

   // loop over size - 1 elements
   for ( int i = 0; i < size - 1; i++ )
   {
      smallest = i; // first index of remaining array

      // loop to find index of smallest element
      for ( int index = i + 1; index < size; index++ )

         if ( array[ index ] < array[ smallest ] )
            smallest = index;

      swap( &array[ i ], &array[ smallest ] );
   } // end if
} // end function selectionSort
```

Declare **array** as **int *array** (rather than **int array[]**) to indicate function **selectionSort** receives single-subscripted array

Receives the size of the array as an argument; declared **const** to ensure that **size** is not modified

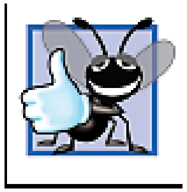
通过前面的例子可知：如果不是以地址作参数，调用函数后得不到交换后的两数

```cpp

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

Arguments are assed by reference, allowing the function to swap values at the original memory locations

```
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
```

6 Selection Sort Using Pass-by-Reference



良好编程习惯： 当要传送一个数组给另一个函数时，一般也将数组大小一起传送。这样可以增加函数的可重用性。

7 sizeof Operators

- `sizeof` operator
  - 返回操作数所占的字节数
  - For arrays, `sizeof` returns

    (size of 1 element) * (number of elements)
  - If `sizeof( int )` returns 4 then
    ```
    int myArray[ 10 ];
    cout << sizeof( myArray );
    ```
    will print 40
  - Can be used with
    - ✓ Variable names
    - ✓ Type names
    - ✓ Constant values

```cpp
// Fig. 8.16: fig08_16.cpp
// Sizeof operator when used on an array name
// returns the number of bytes in the array.
#include <iostream>
using std::cout;
using std::endl;

size_t getSize( double * ); // prototype

int main()
{
   double array[ 20 ]; // 20 doubles; occupies 160 bytes on our system

   cout << "The number of bytes in the array is " << sizeof( array );

   cout << "\nThe number of bytes returned by getSize is "
      << getSize( array ) << endl;
   return 0; // indicates successful termination
} // end main

// return size of ptr
size_t getSize( double *ptr )
{
   return sizeof( ptr );
} // end function getSize
```

Operator **sizeof** applied to an array returns total number of bytes in the array

返回存放address数组的首地址所需字节数

返回该指针所占的字节数

```
The number of bytes in the array is 160
The number of bytes returned by getSize is 4
```

7 sizeof Operators (Cont.)

- `sizeof` operator (Cont.)
  - 在编译阶段完成
  - Eg. `double realArray[ 22 ];`
    - ✓ Use `sizeof realArray / sizeof( double )` to calculate the number of elements in `realArray`
  - Parentheses(括号) are only required if the operand is a type name

```
// Fig. 8.17: fig08_17.cpp
// Demonstrating the sizeof operator.
#include <iostream>
using std::cout;
using std::endl;

int main()
{
   char c; // variable of type char
   short s; // variable of type short
   int i; // variable of type int
   long l; // variable of type long
   float f; // variable of type float
   double d; // variable of type double
   long double ld; // variable of type long double
   int array[ 20 ]; // array of int
   int *ptr = array; // variable of type int *
   cout << "sizeof c = " << sizeof c
      << "\tsizeof(char) = " << sizeof( char )
      << "\nsizeof s = " << sizeof s
      << "\tsizeof(short) = " << sizeof( short )
      << "\nsizeof i = " << sizeof i
      << "\tsizeof(int) = " << sizeof( int )
      << "\nsizeof l = " << sizeof l
      << "\tsizeof(long) = " << sizeof( long )
      << "\nsizeof f = " << sizeof f
      << "\tsizeof(float) = " << sizeof( float )
      << "\nsizeof d = " << sizeof d
      << "\tsizeof(double) = " << sizeof( double )
      << "\nsizeof ld = " << sizeof ld
      << "\tsizeof(long double) = " << sizeof( long double )
      << "\nsizeof array = " << sizeof array
      << "\nsizeof ptr = " << sizeof ptr << endl;
   return 0; // indicates successful termination
} // end main
```

Operator **sizeof** can be used on a variable name

Operator **sizeof** can be used on a type name

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

Operator **sizeof** returns the total number of bytes in the array

7 sizeof Operators (Cont.)

良好编程习惯：数据类型具体需要多大空间进行存储一般会因系统不同而不同，如果采用 sizeof 的方法进行获取可使程序具有更大的可重用性。

错误预防技巧：sizeof运算是在编译阶段进行，而不是在程序运行阶段进行，因此，sizeof的合理使用并不会降低系统的性能。

8 Pointer Expressions and Pointer Arithmetic

- Pointer assignment
  - 同一类型之间的指针可以相互赋值
    - ✓ 如果为不同类型，需要使用类型转换运算符
    - ✓ 例外：void *（代表任何类型）
      - ◇ 无须将指针转换为void *
      - ◇ 需要将void *转换为其他类型
      - ◇ void 指针不能被 dereferenced(绝对不能企图使用该指针所指向的内存中所存储的内容)

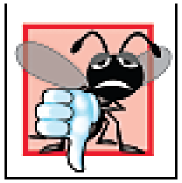
9 Relationship Between Pointers and Arrays

- 数组与指针密切相关
  - 数组名为 constant 指针，表示数组所占内存地址不能改变
  - 指针可以用来进行数组的索引操作

9 Relationship Between Pointers and Arrays

- 使用指针访问数组元素
  - int b[5];
    int *bPtr;
    bPtr = b;
  - b[n] = *(bPtr + n)
  - &b[3] = bPtr + 3（向后跳3个元素，不能理解为3字节）
  - b[3] = *(b + 3)
  - b[3] = bPtr[3]

9 Relationship Between Pointers and Arrays



常见编程错误：尽管数组名是指向数组开头的指针，并且指针可在算术表达式中修改，但是数组名不可以在算术表达式中修改，因为数组名实际上是个常量指针，永远指向数据的第一个元素。

```
// Fig. 8.20: fig08_20.cpp
// Using subscripting and pointer notations with arrays.
#include <iostream>
using std::cout;
using std::endl;

int main()
{
   int b[] = { 10, 20, 30, 40 }; // create 4-element array b
   int *bPtr = b; // set bPtr to point to array b

   // output array b using array subscript notation
   cout << "Array b printed with:\n\nArray subscript notation\n";

   for ( int i = 0; i < 4; i++ )
      cout << "b[" << i << "] = " << b[ i ] << '\n';

   // output array b using the array name and pointer/offset notation
   cout << "\nPointer/offset notation where "
      << "the pointer is the array name\n";

   for ( int offset1 = 0; offset1 < 4; offset1++ )
      cout << "*(b + " << offset1 << ") = " << *( b + offset1 ) << '\n';
```

Using array subscript notation

Using array name and pointer/offset notation

```

   // output array b using bPtr and array subscript notation
   cout << "\nPointer subscript notation\n";

   for ( int j = 0; j < 4; j++ )
      cout << "bPtr[" << j << "] = " << bPtr[ j ] << '\n';

   cout << "\nPointer/offset notation\n";

   // output array b using bPtr and pointer/offset notation
   for ( int offset2 = 0; offset2 < 4; offset2++ )
      cout << "*(bPtr + " << offset2 << ") = "
         << *( bPtr + offset2 ) << '\n';

   return 0; // indicates successful termination
} // end main
```

Using pointer subscript notation

Using pointer name and pointer/offset notation

```
Array b printed with:

Array subscript notation
b[0] = 10
b[1] = 20
b[2] = 30
b[3] = 40

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

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

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

C++ How to Program

```
// Fig. 8.21: fig08_21.cpp
// Copying a string using array notation and pointer notation.
#include <iostream>
using std::cout;
using std::endl;

void copy1( char *, const char * ); // prototype.前面是目的，后面是源
void copy2( char *, const char * ); // prototype。把后面的内容复制到前面

int main()
{
   char string1[ 10 ];
   char *string2 = "Hello";
   char string3[ 10 ];
   char string4[] = "Good Bye";

   copy1( string1, string2 ); // copy string2 into string1
   cout << "string1 = " << string1 << endl;

   copy2( string3, string4 ); // copy string4 into string3
   cout << "string3 = " << string3 << endl;
   return 0; // indicates successful termination
} // end main
```

```
// copy s2 to s1 using array notation
void copy1( char * s1, const char * s2 )
{
   // copying occurs in the for header
   for ( int i = 0; ( s1[ i ] = s2[ i ] ) != '\0'; i++ )
      ; // do nothing in body
} // end function copy1

// copy s2 to s1 using pointer notation
void copy2( char *s1, const char *s2 )
{
   // copying occurs in the for header
   for ( ; ( *s1 = *s2 ) != '\0'; s1++, s2++ )
      ; // do nothing in body
} // end function copy2
```

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

通过数组下标符号将 string **s2** 复制到 **s1**

通过指针符号将s2复制到s1

Increment both pointers to point to next elements in corresponding arrays

我们能够写出这样精巧的小程序吗？

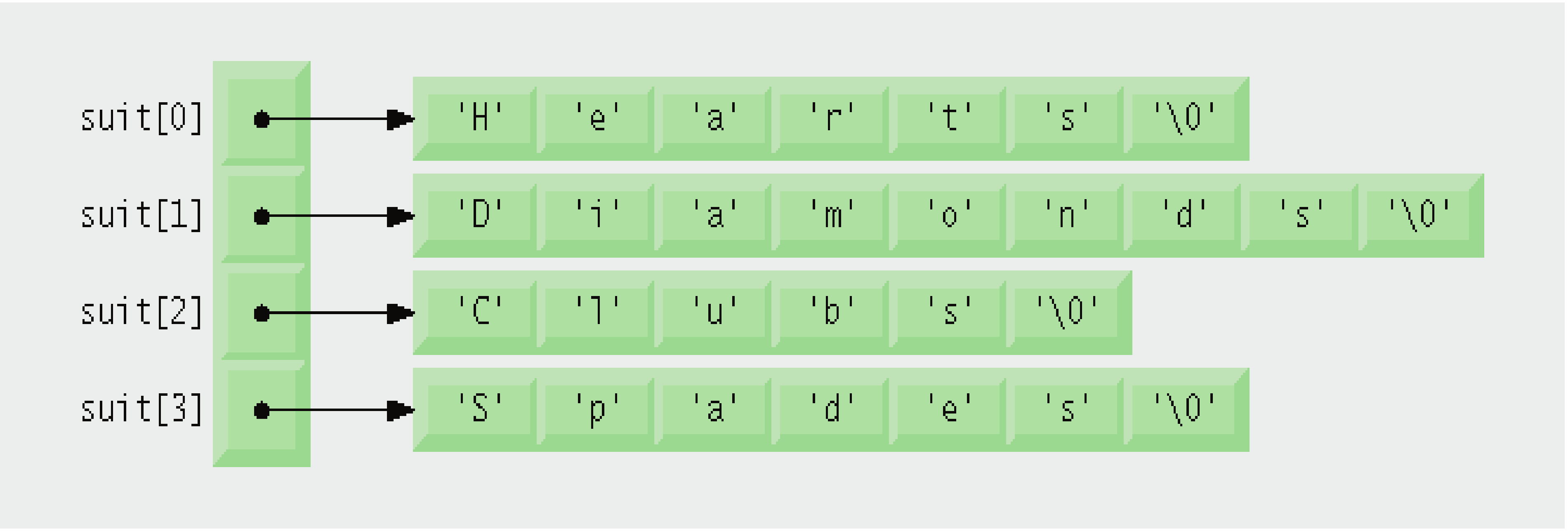
10 Arrays of Pointers

- 数组中可包含指针
  - 通常用来存储字符串数组
    - ✓ 例如：

      const char *suit[4] =

      { "Hearts", "Diamonds", "Clubs", "Spades" };
    - ✓ suit 数组元素具有固定长度(**每个字符串的首地址**)，但其指向的字符串可以为任意长度
    - ✓ 数组存放的是字符串首地址，而不是字符串本身

10 Arrays of Pointers


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

11 Function Pointers

- 函数指针
  - 包含函数的地址
    - ✓ 函数名为函数的起始地址

```
// Fig. 8.28: fig08_28.cpp
// Multipurpose sorting program using function pointers.
#include <iostream>
using std::cout;
using std::cin;
using std::endl;

#include <iomanip>
using std::setw;

// prototypes
void selectionSort( int [], const int, bool (*)( int, int ) );
void swap( int * const, int * const );
bool ascending( int, int ); // implements ascending order
bool descending( int, int ); // implements descending order

int main()
{
   const int arraySize = 10;
   int order; // 1 = ascending, 2 = descending
   int counter; // array index
   int a[ arraySize ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };

   cout << "Enter 1 to sort in ascending order,\n"
      << "Enter 2 to sort in descending order: ";
   cin >> order;
   cout << "\nData items in original order\n";
```

Parameter is pointer to function that receives two integer parameters and returns **bool** result
一个指向函数的指针，该函数有两个输入参数，返回值是bool型

```
   // output original array
   for ( counter = 0; counter < arraySize; counter++ )
      cout << setw( 4 ) << a[ counter ];

   // sort array in ascending order; pass function ascending
   // as an argument to specify ascending sorting order
   if ( order == 1 )
   {
      selectionSort( a, arraySize, ascending );
      cout << "\nData items in ascending order\n";
   } // end if

   // sort array in descending order; pass function descending
   // as an argument to specify descending sorting order
   else
   {
      selectionSort( a, arraySize, descending );
      cout << "\nData items in descending order\n";
   } // end else part of if...else

   // output sorted array
   for ( counter = 0; counter < arraySize; counter++ )
      cout << setw( 4 ) << a[ counter ];

   cout << endl;
   return 0; // indicates successful termination
} // end main
```

Pass pointers to functions **`ascending`** and **`descending`** as parameters to function **`selectionSort`**

```

// multipurpose selection sort; the parameter compare is a pointer to
// the comparison function that determines the sorting order
void selectionSort( int work[], const int size,
                    bool (*compare)( int, int ) )
{
   int smallestOrLargest; // index of smallest (or largest) element

   // loop over size - 1 elements
   for ( int i = 0; i < size - 1; i++ )
   {
      smallestOrLargest = i; // first index of remaining ve

      // loop to find index of smallest (or largest) elemen
      for ( int index = i + 1; index < size; index++ )
         if ( !(*compare)( work[ smallestOrLargest ], work[ index ] ) )
            smallestOrLargest = index;

      swap( &work[ smallestOrLargest ], &work[ i ] );
   } // end if
} // end function selectionSort

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

compare 是一个指向函数的指针，该函数返回一个 **bool** 值

必须用括号括起来，表示是函数的指针

Dereference pointer **compare** to execute the function

```cpp

// determine whether element a is less than
// element b for an ascending order sort
bool ascending( int a, int b )
{
   return a < b; // returns true if a is less than b
} // end function ascending

// determine whether element a is greater than
// element b for a descending order sort
bool descending( int a, int b )
{
   return a > b; // returns true if a is greater than b
} // 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
```

11 Function Pointers (Cont.)

- 函数指针数组
  - 指：一个数组，其元素是一组地址，每个地址指向一个函数
  - 函数名与数组名类似，表示该函数在内存中的首地址
  - 常用于一些菜单驱动的应用程序中

```cpp
// Fig. 8.29: fig08_29.cpp
// Demonstrating an array of pointers to functions.
#include <iostream>
using std::cout;
using std::cin;
using std::endl;

// function prototypes -- each function performs similar actions
void function0( int );
void function1( int );
void function2( int );

int main()
{
   // initialize array of 3 pointers to functions that each
   // take an int argument and return void
   void (*f[ 3 ])( int ) = { function0, function1, function2 };

   int choice;

   cout << "Enter a number between 0 and 2, 3 to end: ";
   cin >> choice;
```

Array initialized with names of three functions

```cpp

   // process user's choice
   while ( ( choice >= 0 ) && ( choice < 3 ) )
   {
      // invoke the function at location choice in
      // the array f and pass choice as an argument
      (*f[ choice ])( choice );

      cout << "Enter a number between 0 and 2, 3 to end: ";
      cin >> choice;
   } // end while

   cout << "Program execution completed." << endl;
   return 0; // indicates successful termination
} // end main

void function0( int a )
{
   cout << "You entered " << a << " so function0 was called\n\n";
} // end function function0

void function1( int b )
{
   cout << "You entered " << b << " so function1 was called\n\n";
} // end function function1
```

Call chosen function by dereferencing corresponding element in array

```cpp

void function2( int c )
{
   cout << "You entered " << c << " so function2 was called\n\n";
} // end function function2
```

```
Enter a number between 0 and 2, 3 to end: 0
You entered 0 so function0 was called

Enter a number between 0 and 2, 3 to end: 1
You entered 1 so function1 was called

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

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

12 String Manipulation Functions

- <cstring>
  - 操纵字符串数据
  - 比较字符串
  - 字符和字符串查找
  - 字符串分隔

(1) 字符和基于指针的字符串

- Character constant
  - Integer value represented as character in single quotes
    - ✓ Example

      `'z'` is integer value of z

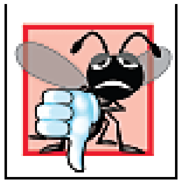
      122 in ASCII

      `'\n'` is integer value of newline

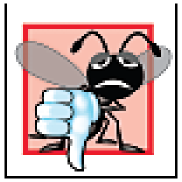
      10 in ASCII

- String
  - ➢ Series of characters treated as single unit
  - ➢ Can include letters, digits, special characters `+, -, *, ...`
  - ➢ String literal (string constants)
    - ✓ Enclosed in double quotes, for example:
      `"I like C++“`
    - ✓ Have `static` storage class
  - ➢ Array of characters, ends with null character `'\0'`
  - ➢ String is constant pointer
    - ✓ Pointer to string’s first character
      Like arrays

- String assignment
  - Character array
    - `char color[] = "blue";`
      - Creates 5 element `char` array `color`
        - Last element is `'\0'`
  - Alternative for character array
    - `char color[] = { 'b', 'l', 'u', 'e', '\0' };`
  - Variable of type `char *`
    - `char *colorPtr = "blue";`
      - Creates pointer `colorPtr` to letter b in string `"blue"`
        - `"blue"` somewhere in memory



常见编程错误：如果不分配足够的内存空间来保存最后的表示结束的空字符，会引起错误。



常见编程错误：如果用数组来保存字符串，要保证数组足够的大从而可以存放最大的字符串，否则，被没有保存完全的内容会被保存到数组后面的内存中，从而引起逻辑错误。

- Reading strings
  - Assign input to character array `word[ 20 ]`
    - `cin >> word;`
      - Reads characters until whitespace or EOF
  - String could exceed array size
    - `cin >> setw( 20 ) >> word;`
      - Reads only up to 19 characters (space reserved for `'\0'`)

- `cin.getline`
  - Read line of text
    - `cin.getline( array, size, delimiter );`
      - Copies input into specified `array` until either
        - One less than `size` is reached
        - `delimiter` character is input
    - Example

      -
        ```
        char sentence[ 80 ];
        cin.getline( sentence, 80, '\n' );
        ```

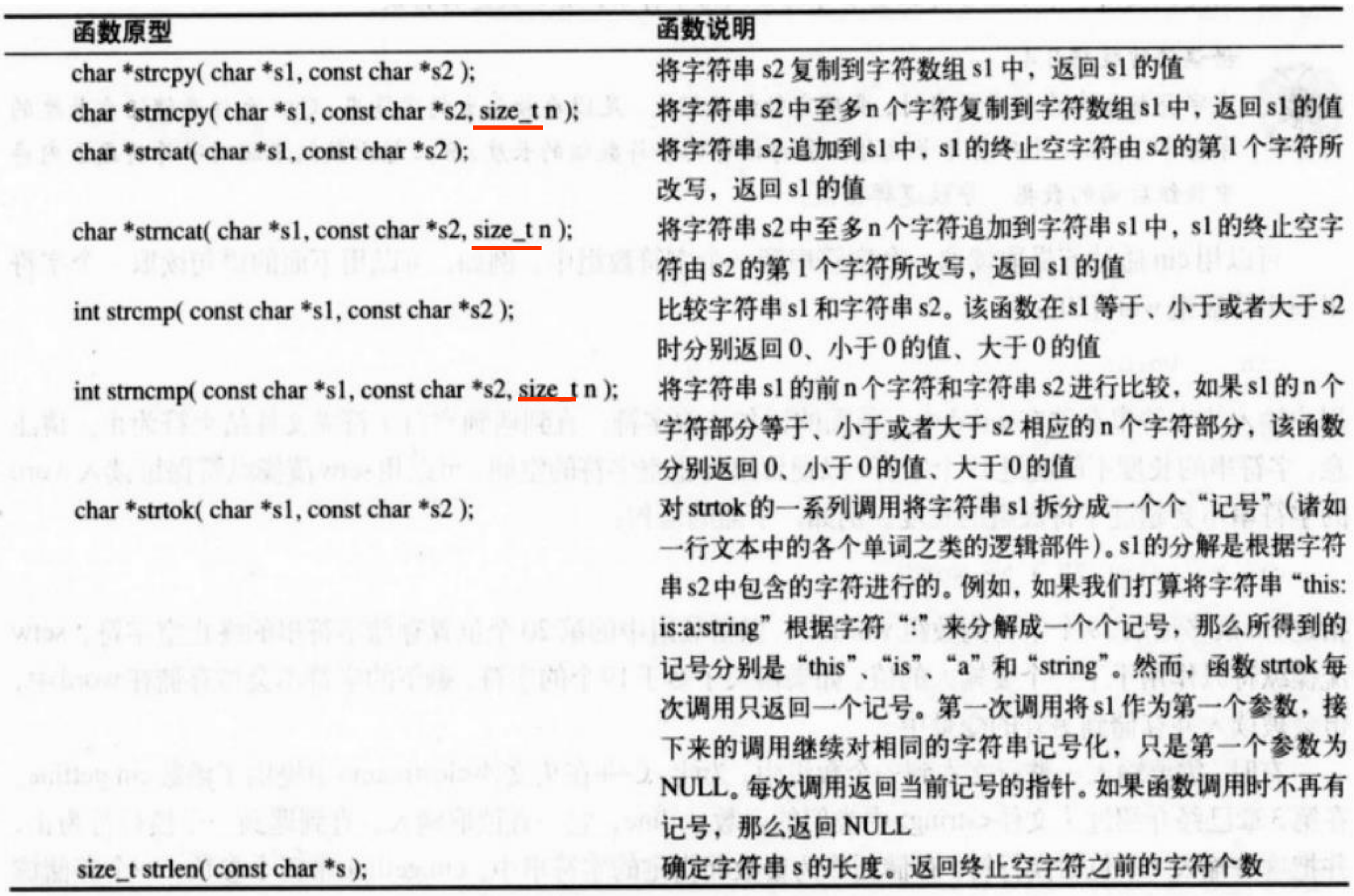
常见编程错误：如果将一个字符采用 char *string的格式表示，会引起 “a fatal runtime”错误。因为char *string表示一个指针，可能代表一个很大的整数，而字符只用很小的整数来表达(ASCII 0~255)，两者之间的差异可能引起操作系统的错误。

常见编程错误：如果将字符串作为参数传送给一个只需要一个字符作为参数的函数时，会引起编译错误。

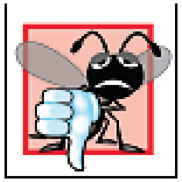
(2) 字符串处理库中的字符串操作函数

- String handling library <`cstring`> provides functions to
  - Manipulate string data
  - Compare strings
  - Search strings for characters and other strings
  - Tokenize strings (separate strings into logical pieces)
- Data type `size_t`
  - Defined to be an unsigned integral type
    - ✓ Such as `unsigned int` or `unsigned long`
  - In header file <`cstring`>

C++ How to Program



函数原型	函数说明
char *strcpy(char *s1, const char *s2);	将字符串 s2 复制到字符数组 s1 中，返回 s1 的值
char *strncpy(char *s1, const char *s2, size_t n);	将字符串 s2 中至多 n 个字符复制到字符数组 s1 中，返回 s1 的值
char *strcat(char *s1, const char *s2);	将字符串 s2 追加到 s1 中，s1 的终止空字符由 s2 的第 1 个字符所改写，返回 s1 的值
char *strncat(char *s1, const char *s2, size_t n);	将字符串 s2 中至多 n 个字符追加到字符串 s1 中，s1 的终止空字符由 s2 的第 1 个字符所改写，返回 s1 的值
int strcmp(const char *s1, const char *s2);	比较字符串 s1 和字符串 s2。该函数在 s1 等于、小于或者大于 s2 时分别返回 0、小于 0 的值、大于 0 的值
int strncmp(const char *s1, const char *s2, size_t n);	将字符串 s1 的前 n 个字符和字符串 s2 进行比较，如果 s1 的 n 个字符部分等于、小于或者大于 s2 相应的 n 个字符部分，该函数分别返回 0、小于 0 的值、大于 0 的值
char *strtok(char *s1, const char *s2);	对 strtok 的一系列调用将字符串 s1 拆分成一个个"记号"(诸如一行文本中的各个单词之类的逻辑部件)。s1 的分解是根据字符串 s2 中包含的字符进行的。例如，如果我们打算将字符串"this:is:a:string"根据字符":"来分解成一个个记号，那么所得到的记号分别是"this"、"is"、"a"和"string"。然而，函数 strtok 每次调用只返回一个记号。第一次调用将 s1 作为第一个参数，接下来的调用继续对相同的字符串记号化，只是第一个参数为 NULL。每次调用返回当前记号的指针。如果函数调用时不再有记号，那么返回 NULL
size_t strlen(const char *s);	确定字符串 s 的长度。返回终止空字符之前的字符个数



常见编程错误：如果没有将头文件<cstring>包含在文件中，采用上面的函数完成相应功能时将出现编译错误。

- 复制字符串
  - `char *strcpy( char *s1, const char *s2 )`
    - ✓ Copies second argument into first argument
      - First argument must be large enough to store string and terminating null character

  -
    ```
    char *strncpy( char *s1, const char *s2,
                   size_t n )
    ```
    - ✓ Specifies number of characters to be copied from second argument into first argument
      - Does not necessarily copy terminating null character

```cpp
// Fig. 8.31: fig08_31.cpp
// Using strcpy and strncpy.
#include <iostream>
using std::cout;
using std::endl;

#include <cstring> // prototypes for strcpy and strncpy
using std::strcpy;
using std::strncpy;

int main()
{
   char x[] = "Happy Birthday to You"; // string length 21
   char y[ 25 ];
   char z[ 15 ];

   strcpy( y, x ); // copy contents of x into y

   cout << "The string in array x is: " << x
      << "\nThe string in array y is: " << y << '\n';
```

`<cstring>` contains prototypes for **`strcpy`** and **`strncpy`**

Copy entire string in array **`x`** into array **`y`**

```
   // copy first 14 characters of x into z
   strncpy( z, x, 14 ); // does not copy null character
   z[ 14 ] = '\0'; // append '\0' to z's contents

   cout << "The string in array z is: " << z << endl;
   return 0; // indicates successful termination
} // end main
```

```
The string in array x is: Happy Birthday to You
The string in array y is: Happy Birthday to You
The string in array z is: Happy Birthday
```

Copy first 14 characters of array **`x`** into array **`z`**. Note that this does not write terminating null character

Append terminating null character

String to copy

Copied string using **`strcpy`**

Copied first 14 characters using **`strncpy`**

- 连接字符串
  - `char *strcat( char *s1, const char *s2 )`
    - ✓ Appends second argument to first argument
      - First character of second argument replaces null character terminating first argument
      - You must ensure first argument large enough to store concatenated result and null character
  - `char *strncat( char *s1, const char *s2,`
    `size_t n )`
    - ✓ Appends specified number of characters from second argument to first argument
      - Appends terminating null character to result

```cpp
// Fig. 8.32: fig08_32.cpp
// Using strcat and strncat.
#include <iostream>
using std::cout;
using std::endl;

#include <cstring> // prototypes for strcat and strncat
using std::strcat;
using std::strncat;

int main()
{
   char s1[ 20 ] = "Happy "; // length 6
   char s2[] = "New Year "; // length 9
   char s3[ 40 ] = "";

   cout << "s1 = " << s1 << "\ns2 = " << s2;

   strcat( s1, s2 ); // concatenate s2 to s1 (length 15)

   cout << "\n\nAfter strcat(s1, s2):\ns1 = " << s1 << "\ns2 = " << s2;

   // concatenate first 6 characters of s1 to s3
   strncat( s3, s1, 6 ); // places '\0' after last character

   cout << "\n\nAfter strncat(s3, s1, 6):\ns1 = " << s1
      << "\ns3 = " << s3;
```

<cstring> contains prototypes for **strcat** and **strncat**

Append **s2** to **s1**

Append first 6 characters of **s1** to **s3**

```

   strcat( s3, s1 ); // concatenate s1 to s3
   cout << "\n\nAfter strcat(s3, s1):\ns1 = " << s1
      << "\ns3 = " << s3 << endl;
   return 0; // indicates successful termination
} // end main
```

Append **s1** to **s3**

```
s1 = Happy
s2 = New Year

After strcat(s1, s2):
s1 = Happy New Year
s2 = New Year

After strncat(s3, s1, 6):
s1 = Happy New Year
s3 = Happy

After strcat(s3, s1):
s1 = Happy New Year
s3 = Happy Happy New Year
```

- 比较字符串
  - `int strcmp( const char *s1, const char *s2 )`
    - ✓ Compares character by character
    - ✓ Returns
      - Zero if strings are equal
      - Negative value if first string is less than second string
      - Positive value if first string is greater than second string
  - `int strncmp( const char *s1, const char *s2, size_t n )`
    - ✓ Compares up to specified number of characters
      - Stops if it reaches null character in one of arguments

```cpp
// Fig. 8.33: fig08_33.cpp
// Using strcmp and strncmp.
#include <iostream>
using std::cout;
using std::endl;

#include <iomanip>
using std::setw;

#include <cstring> // prototypes for strcmp and strncmp
using std::strcmp;
using std::strncmp;

int main()
{
   char *s1 = "Happy New Year";
   char *s2 = "Happy New Year";
   char *s3 = "Happy Holidays";

   cout << "s1 = " << s1 << "\ns2 = " << s2 << "\ns3 = " << s3
      << "\n\nstrcmp(s1, s2) = " << setw( 2 ) << strcmp( s1, s2 )
      << "\nstrcmp(s1, s3) = " << setw( 2 ) << strcmp( s1, s3 )
      << "\nstrcmp(s3, s1) = " << setw( 2 ) << strcmp( s3, s1 );

   cout << "\n\nstrncmp(s1, s3, 6) = " << setw( 2 )
      << strncmp( s1, s3, 6 ) << "\nstrncmp(s1, s3, 7) = " << setw( 2 )
      << strncmp( s1, s3, 7 ) << "\nstrncmp(s3, s1, 7) =
      << strncmp( s3, s1, 7 ) << endl;
   return 0; // indicates successful termination
} // end main
```

<cstring> contains prototypes for **strcmp** and **strncmp**

Compare **s1** and **s2**

Compare **s1** and **s3**

Compare **s3** and **s1**

Compare up to 6 characters of **s1** and **s3**

Compare up to 7 characters of **s1** and **s3**

Compare up to 7 characters of **s3** and **s1**

```
s1 = Happy New Year
s2 = Happy New Year
s3 = Happy Holidays

strcmp(s1, s2) =  0
strcmp(s1, s3) =  1
strcmp(s3, s1) = -1

strncmp(s1, s3, 6) =  0
strncmp(s1, s3, 7) =  1
strncmp(s3, s1, 7) = -1
```

- Tokenizing
  - Breaking strings into tokens
    - ✓ Tokens usually logical units, such as words (separated by spaces)
    - ✓ Separated by delimiting characters
  - Example
    - ✓ `"This is my string"` has 4 word tokens (separated by spaces)

Tokenizing (Cont.)

- `char *strtok( char *s1, const char *s2 )`
  - ✓ Multiple calls required
    - First call contains two arguments, string to be tokenized and string containing delimiting characters
      - Finds next delimiting character and replaces with null character
    - Subsequent calls continue tokenizing
      - Call with first argument `NULL`
      - Stores pointer to remaining string in a `static` variable
  - ✓ Returns pointer to current token

```cpp
// Fig. 8.34: fig08_34.cpp
// Using strtok.
#include <iostream>
using std::cout;
using std::endl;

#include <cstring> // prototype for strtok
using std::strtok;

int main()
{
   char sentence[] = "This is a sentence with 7 tokens";
   char *tokenPtr;

   cout << "The string to be tokenized is:\n" << sentence
      << "\n\nThe tokens are:\n\n";

   // begin tokenization of sentence
   tokenPtr = strtok( sentence, " " );

   // continue tokenizing sentence until tokenPtr becomes NULL
   while ( tokenPtr != NULL )
   {
      cout << tokenPtr << '\n';
      tokenPtr = strtok( NULL, " " ); // get next token
   } // end while

   cout << "\nAfter strtok, sentence = " << sentence << endl;
   return 0; // indicates successful termination
} // end main
```

`<cstring>` contains prototype for **`strtok`**

First call to **`strtok`** begins tokenization

Subsequent calls to **`strtok`** with **NULL** as first argument to indicate continuation

```
The string to be tokenized is:
This is a sentence with 7 tokens

The tokens are:

This
is
a
sentence
with
7
tokens

After strtok, sentence = This
```

- Determining string lengths
  - `size_t strlen( const char *s )`
    - Returns number of characters in string
      - Terminating null character is not included in length
      - This length is also the index of the terminating null character

```cpp
// Fig. 8.35: fig08_35.cpp
// Using strlen.
#include <iostream>
using std::cout;
using std::endl;

#include <cstring> // prototype for strlen
using std::strlen;

int main()
{
   char *string1 = "abcdefghijklmnopqrstuvwxyz";
   char *string2 = "four";
   char *string3 = "Boston";

   cout << "The length of \"" << string1 << "\" is " << strlen( string1 )
      << "\nThe length of \"" << string2 << "\" is " << strlen( string2 )
      << "\nThe length of \"" << string3 << "\" is " << strlen( string3 )
      << endl;
   return 0; // indicates successful termination
} // end main
```

<cstring> contains prototype for **strlen**

Using **strlen** to determine length of strings

```
The length of "abcdefghijklmnopqrstuvwxyz" is 26
The length of "four" is 4
The length of "Boston" is 6
```

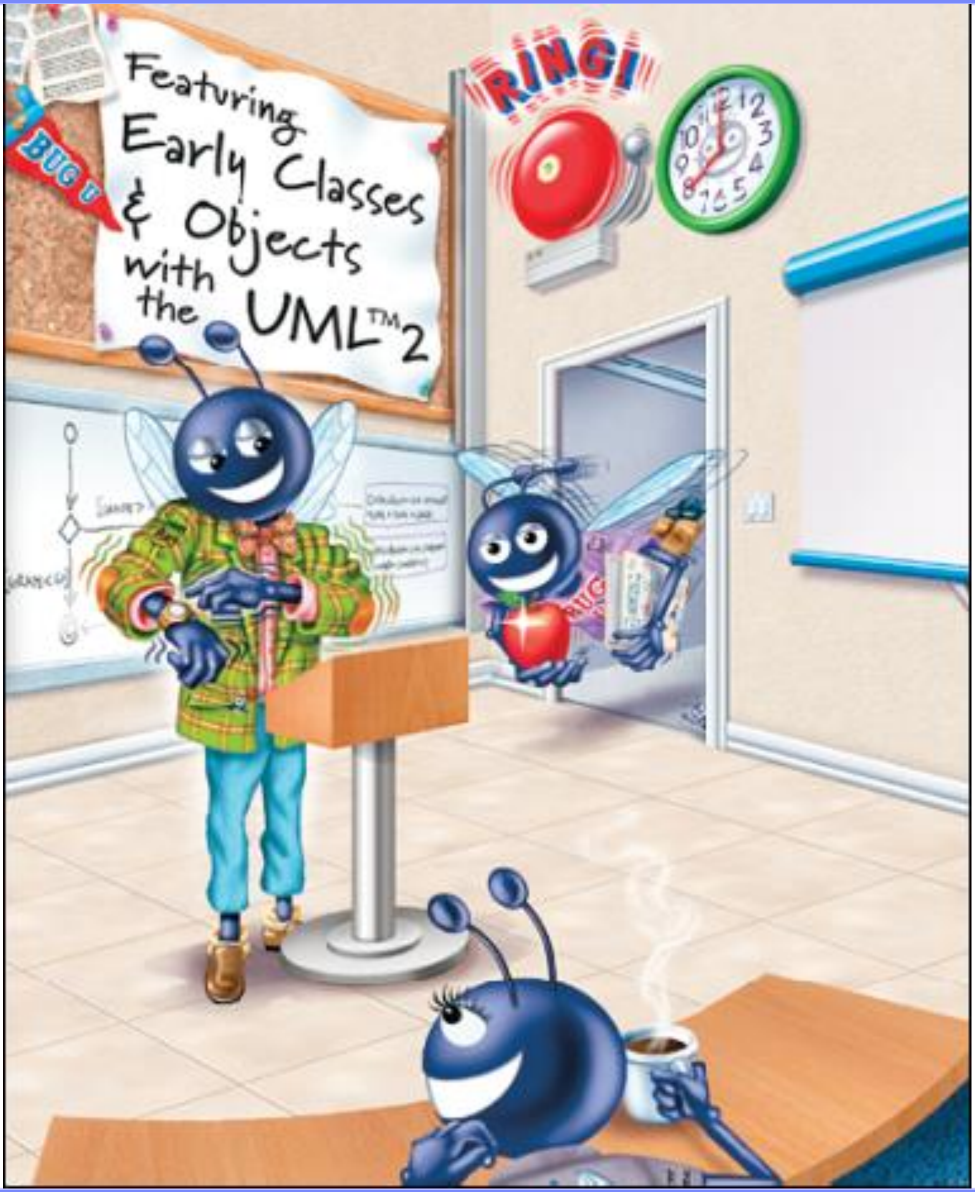
思考题：下面程序的运行结果？为什么？

```
char *GetMemory(void)
{
     char p[] = "hello world";
     return p;
}

int main()
{
    char *str = NULL;

    str = GetMemory();
    cout << str << endl;

    return 0;
}
```

思考题：下面程序的运行结果？为什么？

```
int main()
{
    char *str = (char *) malloc(100);

    strcpy(str, "hello");
    free(str);

    if(str != NULL)
    {
           strcpy(str, "world");
           cout << str << endl;
    }

    return 0;
}
```

C++ How to Program



Thank you!