8

Pointers and Pointer-Based Strings

东南大学软件学院

1

OBJECTIVES

In this chapter you'll learn:

- What pointers are. 什么是指针
- The similarities and differences between pointers and references and when to use each.
- To use pointers to pass arguments to functions (by reference).
- To use pointer-based C-style strings.
- The close relationships among pointers, arrays and C-style strings.
- To use pointers to functions. 函数指针
- To declare and use arrays of C-style strings.

2

8.1 Introduction

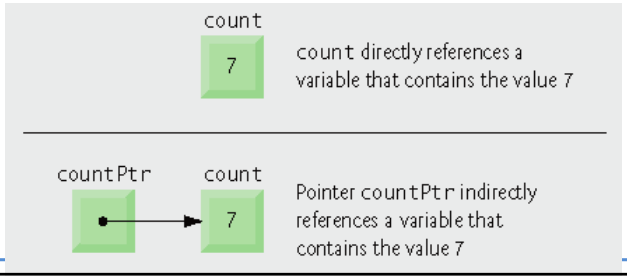
- Pointers 指针
  - Powerful, but difficult to master
  - Can be used to perform pass-by-reference
  - Can be used to create and manipulate dynamic data structures
  - Close relationship with arrays and strings

    `char` * pointer-based strings

3

8.2 Pointer Variable Declarations and Initialization

- Pointer variables
  - Contain memory addresses 内存地址 as values
    - Normally, variable contains specific value (direct reference)
    - Pointers contain address of variable that has specific value (indirect reference)



4

8.2 Pointer Variable Declarations and Initialization (Cont.)

- Pointer declarations 声明
  - * indicates variable is a pointer
    - Example

```
int *myPtr;
```

      - »Declares pointer to `int`, of type `int *`
    - Multiple pointers require multiple asterisks

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

- Pointer initialization 初始化
  - Initialized to `0`, `NULL`, or an address
    - `0` or `NULL` points to nothing (null pointer)

5

Error-Prevention Tip 8.1

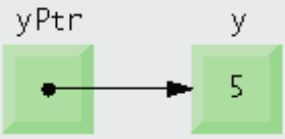
Initialize pointers to prevent pointing to unknown or uninitialized areas of memory.

指针变量在使用之前需要进行初始化。

6

8.3 Pointer Operators

- Address operator (&)地址运算符
  - Returns memory address of its operand
  - Example
    - `int y = 5;`
      `int *yPtr;`
      `yPtr = &y;`
      assigns the address of variable `y` to pointer variable `yPtr`



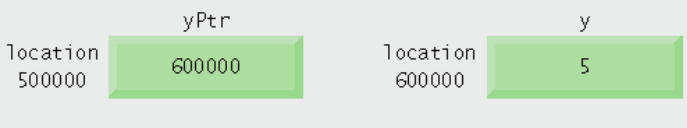
7

8.3 Pointer Operators (Cont.)

- * operator
  - Also called indirection operator or dereferencing operator
    间接（引用）运算符
  - `*yPtr` returns `y` (because `yPtr` points to `y`)
  - Dereferenced pointer is an *lvalue*

```
*yptr = 9;
```

- * and & are inverses of each other
  - Will "cancel one another out" when applied consecutively in either order (*&y, &*yptr)



8

Common Programming Error 8.2

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

9

Portability Tip 8.1

The format in which a pointer is output is compiler dependent. Some systems output pointer values as hexadecimal integers（十六进制数）, some use decimal integers（十进制数） and some use other formats.

地址输出格式与编译器相关

10

```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
   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
```

11

```
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
```

12

Operators	Associativity	Type
`()` `[]`	left to right	highest
`++` `--` `static_cast<` *type* `>(` *operand* `)`	left to right	unary (postfix)
`++` `--` `+` `-` `!` `&` `*`	right to left	unary (prefix)
`*` `/` `%`	left to right	multiplicative
`+` `-`	left to right	additive
`<<` `>>`	left to right	insertion/extraction
`<` `<=` `>` `>=`	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. 8.5 | Operator precedence and associativity.

13

8.4 Passing Arguments to Functions by Reference with Pointers

- Three ways to pass arguments to a function
  - Pass-by-value 传值调用
  - Pass-by-reference with reference arguments 传引用的引用调用
  - Pass-by-reference with pointer arguments 传地址的引用调用
- A function can `return` only one value
- Arguments passed to a function using reference arguments
  - Function can modify original values of arguments
    - More than one value “returned”

14

8.4 Passing Arguments to Functions by Reference with Pointers (Cont.)

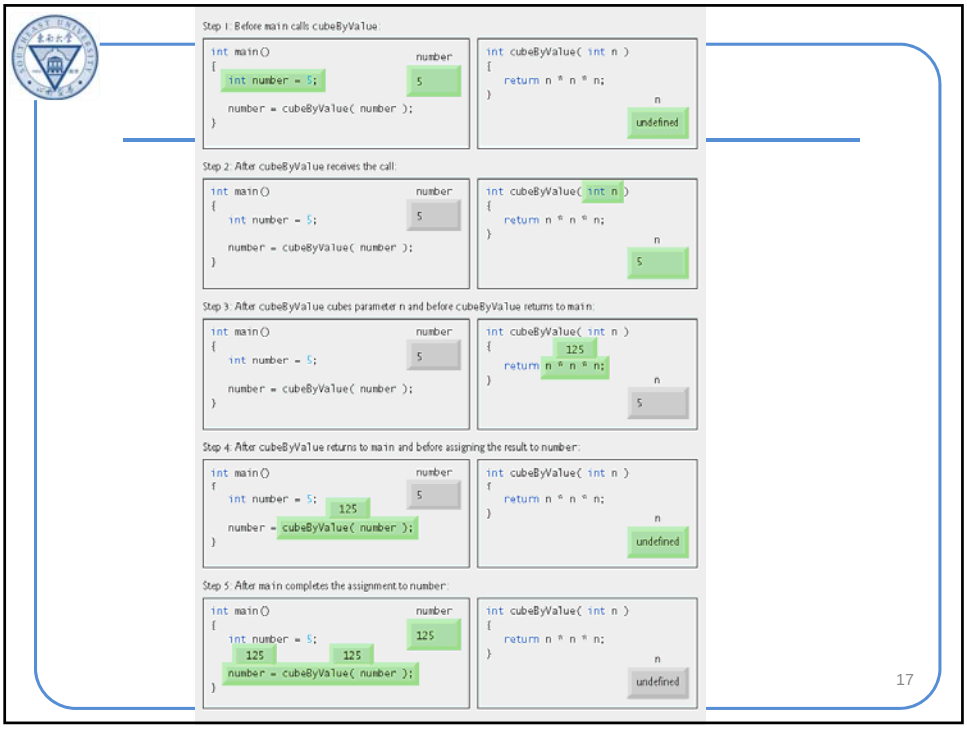
- Pass-by-reference with pointer arguments
  - Simulates pass-by-reference
    - Use pointers and indirection operator
  - Pass address of argument using & operator
  - Arrays not passed with & because array name is already a pointer 数组名不需要地址运算符
  - * operator used as alias/nickname for variable inside of function

15

```
1 // Fig. 8.6: fig08_06.cpp
2 // Cube a variable using pass-by-value.
6
7 int cubeByValue( int ); // prototype
8
9 int main()
10{
11   int number = 5;
12
13   cout << "The original value of number is " << number;
14
15   number = cubeByValue( number ); // pass number by value to cubeByValue
16   cout << "\nThe new value of number is " << number << endl;
17   return 0; // indicates successful termination
18} // end main
19
20// calculate and return cube of integer argument
21int cubeByValue( int n )
22{
23   return n * n * n; // cube local variable n and return result
24} // end function cubeByValue
```

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

16

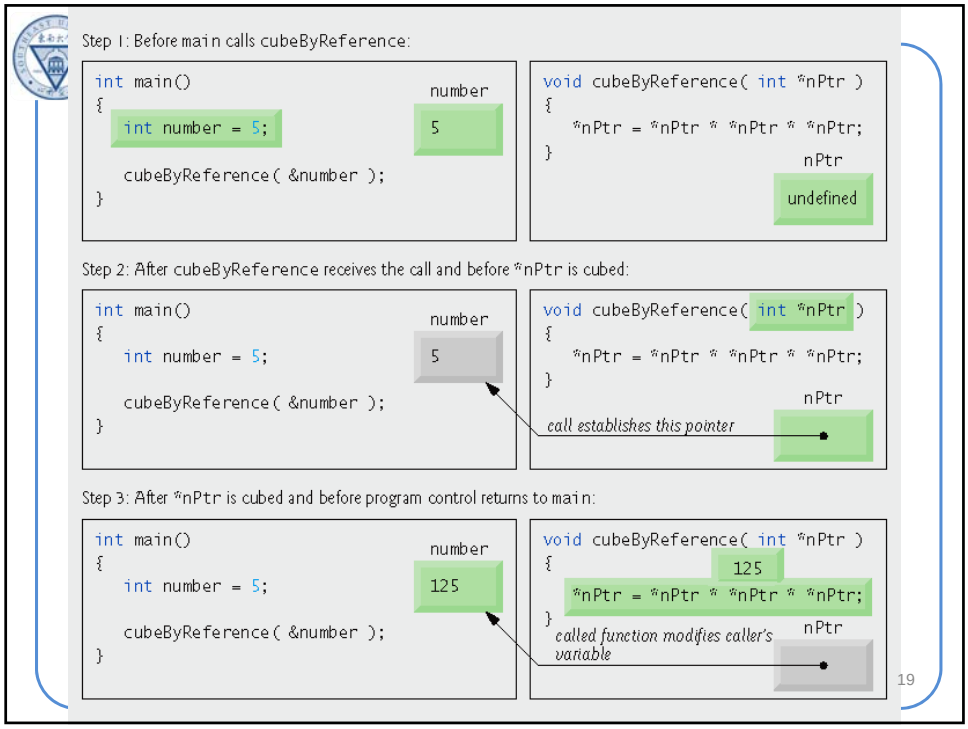


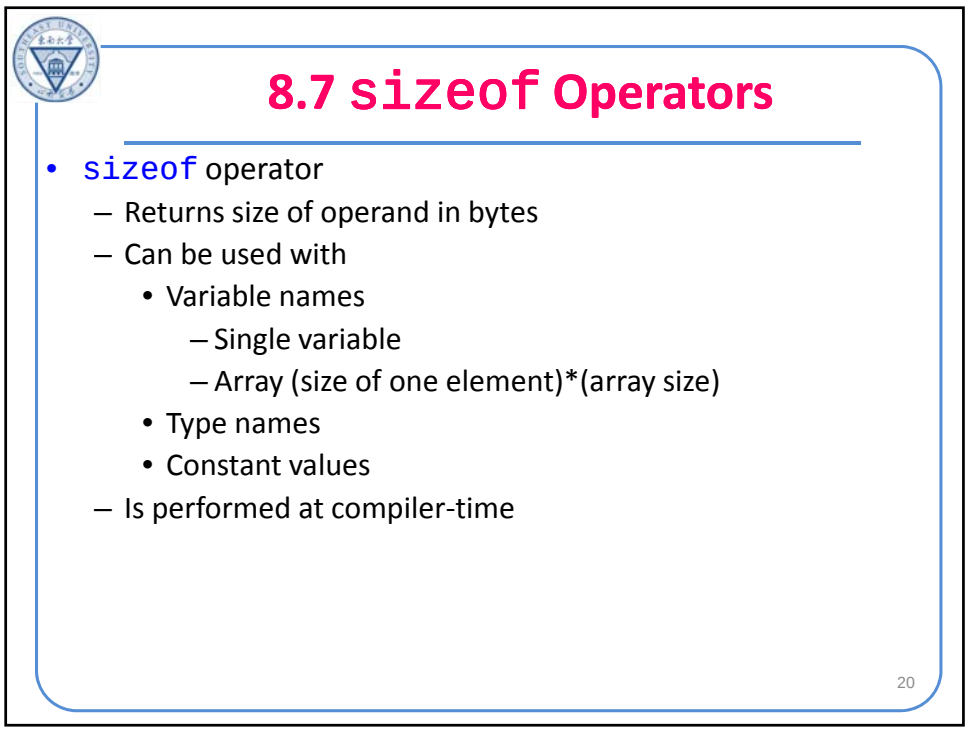
17

```
1 // Fig. 8.7: fig08_07.cpp
2 // Cube a variable using pass-by-reference with a pointer argument.
6
7 void cubeByReference( int * ); // prototype
8
9 int main()
10{
11   int number = 5;
12
13   cout << "The original value of number is " << number;
14
15   cubeByReference( &number ); // pass number address to cubeByReference
16
17   cout << "\nThe new value of number is " << number << endl;
18   return 0; // indicates successful termination
19} // end main
20
21// calculate cube of *nPtr; modifies variable number in main
22void cubeByReference( int *nPtr )
23{
24   *nPtr = *nPtr * *nPtr * *nPtr; // cube *nPtr
25} // end function cubeByReference
```

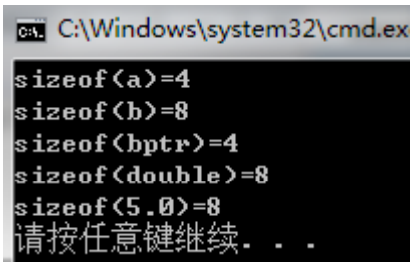
```
The original value of number is 5
The new value of number is 125
```

18


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
19


8.7 sizeof Operators
sizeof operator
Returns size of operand in bytes
Can be used with
Variable names
Single variable
Array (size of one element)*(array size)
Type names
Constant values
Is performed at compiler-time
20

```cpp
#include <iostream>
using namespace std;
int main()
{
        int a=0;
        int b[2];
        int *bptr = b;
        cout<<"sizeof(a)="<<sizeof(a)<<endl;
        cout<<"sizeof(b)="<<sizeof(b)<<endl;
        cout<<"sizeof(bptr)="<<sizeof(bptr)<<endl;
        cout<<"sizeof(double)="<<sizeof(double)<<endl;
        cout<<"sizeof(5.0)="<<sizeof(5.0)<<endl;
}
```



21

```cpp
// 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 *
```

22

```
19   cout << "sizeof c = " << sizeof c
20      << "\tsizeof(char) = " << sizeof( char )
21      << "\nsizeof s = " << sizeof s
22      << "\tsizeof(short) = " << sizeof( short )
23      << "\nsizeof i = " << sizeof i
24      << "\tsizeof(int) = " << sizeof( int )
25      << "\nsizeof l = " << sizeof l
26      << "\tsizeof(long) = " << sizeof( long )
27      << "\nsizeof f = " << sizeof f
28      << "\tsizeof(float) = " << sizeof( float )
29      << "\nsizeof d = " << sizeof d
30      << "\tsizeof(double) = " << sizeof( double )
31      << "\nsizeof ld = " << sizeof ld
32      << "\tsizeof(long double) = " << sizeof( long double )
33      << "\nsizeof array = " << sizeof array
34      << "\nsizeof ptr = " << sizeof ptr << endl;
35   return 0; // indicates successful termination
36} // end main
```

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

23

24

8.8 Pointer Expressions and Pointer Arithmetic

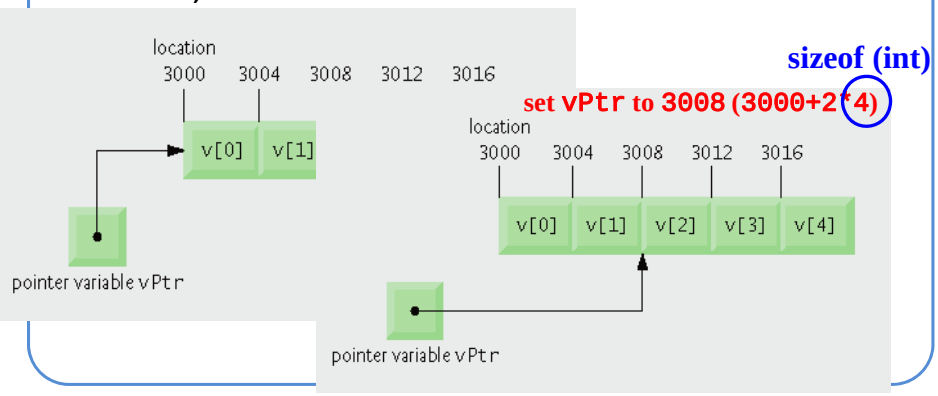
- Pointer arithmetic 指针算术运算
  - Increment/decrement pointer (++ or - -)
  - Add/subtract an integer to/from a pointer (+ or +=, - or -=)
  - Pointers may be subtracted from each other
  - Pointer arithmetic is meaningless unless performed on a pointer to an array

24

8.8 Pointer Expressions and Pointer Arithmetic

```
int v[5];
int *vPtr = v; //vPtr points to first element v[ 0 ]
vPtr += 2;
```

`vPtr = &v[ 0 ];`



8.8 Pointer Expressions and Pointer Arithmetic

- Subtracting pointers

```
vPtr2 = &v[ 2 ];
vPtr = &v[ 0 ];
cout << vPtr2 - vPtr;
```

Common Programming Error 8.11

Using pointer arithmetic to increment or decrement a pointer such that the pointer refers to an element past the end of the array or before the beginning of the array is normally a logic error.使用指针算术运算操作数组指针要防止越界。

8.8 Pointer Expressions and Pointer Arithmetic

- Pointer assignment 指针赋值
  - Pointer can be assigned to another pointer if both are of same type
    - If not same type, cast operator must be used
    - Exception
      - Pointer to `void` (type `void *`)
        - » Generic pointer, represents any type 一般性指针
        - » No casting needed to convert pointer to `void *`
        - » Casting is needed to convert `void *` to any other type
        - » `void` pointers cannot be dereferenced

8.8 Pointer Expressions and Pointer Arithmetic

- Pointer comparison 指针比较
  - Use equality (==) and relational (>, <, >=, <=) operators
  - Compare addresses stored in pointers 比较内存地址
    - Comparisons are meaningless unless pointers point to members of the same array
  - Commonly used to determine whether pointer is 0 (null pointer)

```
if (ptr != NULL)
```

8.9 Relationship Between Pointers and Arrays

- Arrays and pointers are closely related
  - Array name is like constant pointer
  - Pointers can do array subscripting operations

```
int b[ 5 ];
int *bPtr;
bPtr = b;
```

```
b[ n ] ⇔ *( bPtr + n )
b[ n ] ⇔ *( b + n )
b[ n ] ⇔ bPtr[ n ]
b+=2
```



```
1 // Fig. 8.20: fig08_20.cpp
2 // Using subscripting and pointer notations with arrays.
6
7 int main()
8 {
9    int b[] = { 10, 20, 30, 40 }; // create 4-element array b
10   int *bPtr = b; // set bPtr to point to array b
11
12   // output array b using array subscript notation
13   cout << "Array b printed with:\n\nArray subscript notation\n";
14
15   for ( int i = 0; i < 4; i++ )
16      cout << "b[" << i << "] = " << b[ i ] << '\n';
17
18   // output array b using the array name and pointer/offset notation
19   cout << "\nPointer/offset notation where "
20      << "the pointer is the array name\n";
21
22   for ( int offset1 = 0; offset1 < 4; offset1++ )
23      cout << "*(b + " << offset1 << ") = " << *( b + offset1 ) << '\n';
```

```
24
25   // output array b using bPtr and array subscript notation
26   cout << "\nPointer subscript notation\n";
27
28   for ( int j = 0; j < 4; j++ )
29      cout << "bPtr[" << j << "] = " << bPtr[ j ] << '\n';
30
31   cout << "\nPointer/offset notation\n";
32
33   // output array b using bPtr and pointer/offset notation
34   for ( int offset2 = 0; offset2 < 4; offset2++ )
35      cout << "*(bPtr + " << offset2 << ") = "
36         << *( bPtr + offset2 ) << '\n';
37
38   return 0; // indicates successful termination
39} // end main
```

31

```
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
```

32

```
1 // Fig. 8.21: fig08_21.cpp
2 // Copying a string using array notation and pointer notation.
6
7 void copy1( char *, const char * ); // prototype
8 void copy2( char *, const char * ); // prototype
9
10int main()
11{
12   char string1[ 10 ];
13   char *string2 = "Hello";
14   char string3[ 10 ];
15   char string4[] = "Good Bye";
16
17   copy1( string1, string2 ); // copy string2 into string1
18   cout << "string1 = " << string1 << endl;
19
20   copy2( string3, string4 ); // copy string4 into string3
21   cout << "string3 = " << string3 << endl;
22   return 0; // indicates successful termination
23} // end main
```

33

```
24
25// copy s2 to s1 using array notation
26void copy1( char * s1, const char * s2 )
27{
28   // copying occurs in the for header
29   for ( int i = 0; ( s1[ i ] = s2[ i ] ) != '\0'; i++ )
30      ; // do nothing in body
31} // end function copy1
32
33// copy s2 to s1 using pointer notation
34void copy2( char *s1, const char *s2 )
35{
36   // copying occurs in the for header
37   for ( ; ( *s1 = *s2 ) != '\0'; s1++, s2++ )
38      ; // do nothing in body
39} // end function copy2
```

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

34

8.5 Using const with Pointers (Cont.)

- Four ways to pass pointer to function
  - Nonconstant pointer to nonconstant data 指向非常量数据的非常量指针
    - Highest amount of access
    - Data can be modified through the dereferenced pointer
    - Pointer can be modified to point to other data
      - Pointer arithmetic
        - » Operator ++ moves array pointer to the next element
    - Its declaration does not include `const` qualifier

35

8.5 Using const with Pointers

- `const` qualifier
  - Indicates that value of variable should not be modified
  - `const` used when function does not need to change the variable's value
- Principle of least privilege
  - Award function enough access to accomplish task, but no more
  - Example
    - A function that prints the elements of an array, takes array and `int` indicating length
      - Array contents are not changed – should be `const`
      - Array length is not changed – should be `const`

36

8.5 Using const with Pointers (Cont.)

- Four ways to pass pointer to function (Cont.)
  - Nonconstant pointer to constant data指向常量数据的非常量指针 const char *cptr;
    - Pointer can be modified to point to any appropriate data item
    - Data cannot be modified through this pointer
    - Provides the performance of pass-by-reference and the protection of pass-by-value

37

```
1 // Fig. 8.11: fig08_11.cpp
2 // Printing a string one character at a time using
3 // a non-constant pointer to constant data.
7
8 void printCharacters( const char * ); // print using pointer to const data
9
10int main()
11{
12   const char phrase[] = "print characters of a string";
13
14   cout << "The string is:\n";
15   printCharacters( phrase ); // print characters in phrase
16   cout << endl;
17   return 0; // indicates successful termination
18} // end main
19
20// sPtr can be modified, but it cannot modify the character to which
21// it points, i.e., sPtr is a "read-only" pointer
22void printCharacters( const char *sPtr )
23{
24   for ( ; *sPtr != '\0'; sPtr++ ) // no initialization
25      cout << *sPtr; // display character without modification
26} // end function printCharacters
```

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

38

```
1 // Fig. 8.12: fig08_12.cpp
2 // Attempting to modify data through a
3 // non-constant pointer to constant data.
4
5 void f( const int * ); // prototype
6
7 int main()
8 {
9    int y;
10
11   f( &y ); // f attempts illegal modification
12   return 0; // indicates successful termination
13} // end main
14
15// xPtr cannot modify the value of constant variable to which it points
16void f( const int *xPtr )
17{
18   *xPtr = 100; // error: cannot modify a const object
19} // end function f
```

Microsoft Visual C++ compiler error message:

```
c:\cpphtp6_examples\ch08\Fig08_12\fig08_12.cpp(18) :
   error C3892: 'xPtr' : you cannot assign to a variable that is const
```

39

8.5 Using const with Pointers (Cont.)

- Four ways to pass pointer to function (Cont.)
  - Constant pointer to nonconstant data 指向非常量数据的常量指针 int* const xptr = x; // int x[5];
    - Always points to the same memory location
      - Can only access other elements using subscript notation *(xptr + 1)
    - Data can be modified through the pointer
    - Default for an array name 数组名是默认情况
    - Must be initialized when declared

40

```
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 be initialized
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
```

Microsoft Visual C++ compiler error message:

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

41

8.5 Using const with Pointers (Cont.)

- Four ways to pass pointer to function (Cont.)
  - Constant pointer to constant data 指向常量数据的常量指针
    const int* const ptr = &x;
    - Least amount of access
    - Always points to the same memory location
    - Data cannot be modified using this pointer

42

```
1 // Fig. 8.14: fig08_14.cpp
2 // Attempting to modify a constant pointer to constant data.
6
7 int main()
8 {
9    int x = 5, y;
10
11   // ptr is a constant pointer to a constant integer.
12   // ptr always points to the same location; the integer
13   // at that location cannot be modified.
14   const int *const ptr = &x;
15
16   cout << *ptr << endl;
17
18   *ptr = 7; // error: *ptr is const; cannot assign new value
19   ptr = &y; // error: ptr is const; cannot assign new address
20   return 0; // indicates successful termination
21} // end 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
```

43

8.6 Selection Sort Using Pass-by-Reference

- Implement `selectionSort` using pointers
  - Selection sort algorithm 选择排序
    - 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 &

44

```
1 // Fig. 8.15: fig08_15.cpp
2 // This program puts values into an array, sorts the values into
3 // ascending order and prints the resulting array.
10
11void selectionSort( int * const, const int ); // prototype
12void swap( int * const, int * const ); // prototype
13
14int main()
15{
16   const int arraySize = 10;
17   int a[ arraySize ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
18
19   cout << "Data items in original order\n";
20
21   for ( int i = 0; i < arraySize; i++ )
22      cout << setw( 4 ) << a[ i ];
23
24   selectionSort( a, arraySize ); // sort the array
25
26   cout << "\nData items in ascending order\n";
27
28   for ( int j = 0; j < arraySize; j++ )
29      cout << setw( 4 ) << a[ j ];
```

45

```
30
31   cout << endl;
32   return 0; // indicates successful termination
33} // end main
34
35// function to sort an array
36void selectionSort( int * const array, const int size )
37{
38   int smallest; // index of smallest element
39
40   // loop over size - 1 elements
41   for ( int i = 0; i < size - 1; i++ )
42   {
43      smallest = i; // first index of remaining array
44
45      // loop to find index of smallest element
46      for ( int index = i + 1; index < size; index++ )
47
48         if ( array[ index ] < array[ smallest ] )
49            smallest = index;
50
51      swap( &array[ i ], &array[ smallest ] );
52   } // end if
53} // end function selectionSort
```

46

```
54
55// swap values at memory locations to which
56// element1Ptr and element2Ptr point
57void swap( int * const element1Ptr, int * const element2Ptr )
58{
59   int hold = *element1Ptr;
60   *element1Ptr = *element2Ptr;
61   *element2Ptr = hold;
62} // end function swap
```

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

47

8.13.1 Fundamentals of Characters and Pointer-Based Strings (Cont.)

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

48

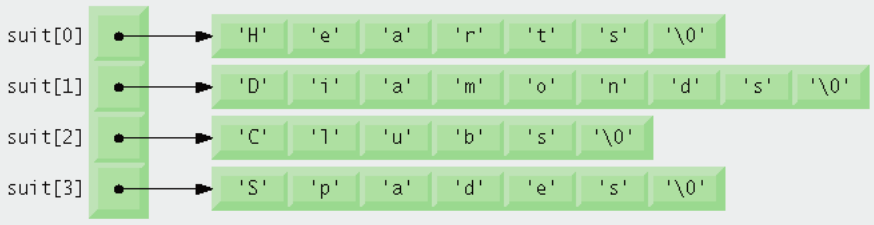
8.10 Arrays of Pointers

- Arrays can contain pointers
  - Commonly used to store array of strings (string array)
    - Array does not store strings, only pointers to strings
    - Example

```
const char *suit[ 4 ] =
  { "Hearts", "Diamonds", "Clubs",
  "Spades" };
```
（花色：红心，方片，梅花，黑桃）

      » Each element of `suit` points to a `char *` (string)
    - `suit` array has fixed size (4), but strings can be of any size

49



50

8.11 Case Study: Card Shuffling and Dealing Simulation

- Card shuffling 洗牌 and dealing 发牌

51

8.11 Case Study: Card Shuffling and Dealing Simulation

- Card shuffling 洗牌 and dealing 发牌

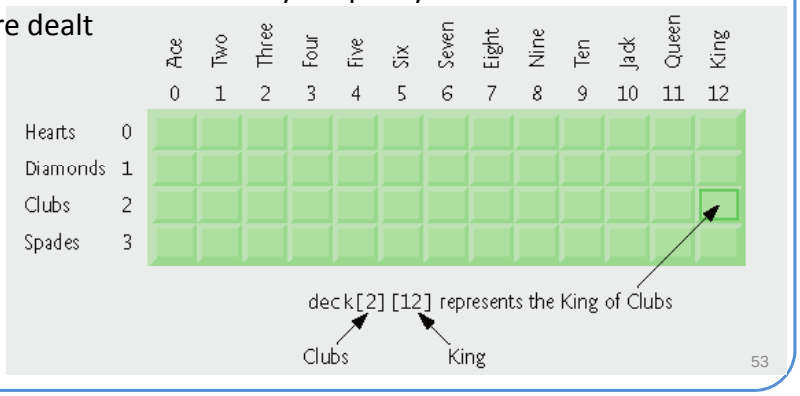
```
 Nine of Spades          Seven of Clubs
  Five of Spades         Eight of Clubs
Queen of Diamonds        Three of Hearts
 Jack of Spades           Five of Diamonds
 Jack of Diamonds        Three of Diamonds
Three of Clubs             Six of Clubs
  Ten of Clubs           Nine of Diamonds
  Ace of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

文字版洗牌程序

52

8.11 Case Study: Card Shuffling and Dealing Simulation

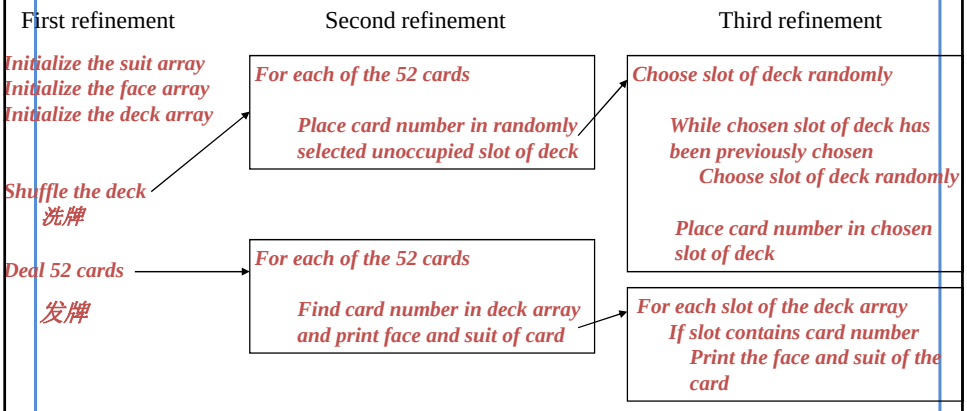
- Card shuffling program
  - Use an array of pointers to strings, to store suit names
  - Use a double scripted array (suit-by-value) 二维数组
  - Place 1-52 into the array to specify the order in which the cards are dealt



53

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

- Pseudocode for shuffling and dealing simulation



54

```
1  Initialize the suit array
2  Initialize the face array
3  Initialize the deck array
4
5  For each of the 52 cards
6     Choose slot of deck randomly
7
8     While slot of deck has been previously chosen
9        Choose slot of deck randomly
10
11    Place card number in chosen slot of deck
12
13 For each of the 52 cards
14    For each slot of deck array
15       If slot contains desired card number
16          Print the face and suit of the card
```

55

```
1  // Fig. 8.25: DeckOfCards.h
2  // Definition of class DeckOfCards that
3  // represents a deck of playing cards.
4
5  // DeckOfCards class definition
6  class DeckOfCards
7  {
8  public:
9     DeckOfCards(); // constructor initializes deck
10    void shuffle(); // shuffles cards in deck
11    void deal(); // deals cards in deck
12 private:
13    int deck[ 4 ][ 13 ]; // represents deck of cards
14 }; // end class DeckOfCards
```

56

```
21// DeckOfCards default constructor initializes deck
22DeckOfCards::DeckOfCards()
23{
24   // loop through rows of deck
25   for ( int row = 0; row <= 3; row++ )
26   {
27      // loop through columns of deck for current row
28      for ( int column = 0; column <= 12; column++ )
29      {
30         deck[ row ][ column ] = 0; // initialize slot of deck to 0
31      } // end inner for
32   } // end outer for
34   srand( time( 0 ) ); // seed random number generator
35} // end DeckOfCards default constructor
36
37// shuffle cards in deck
38void DeckOfCards::shuffle()
39{
40   int row; // represents suit value of card
41   int column; // represents face value of card
43   // for each of the 52 cards, choose a slot of the deck randomly
44   for ( int card = 1; card <= 52; card++ )
45   {
46      do // choose a new random location until unoccupied slot is found
47      {
48         row = rand() % 4; // randomly select the row
49         column = rand() % 13; // randomly select the column
50      } while( deck[ row ][ column ] != 0 ); // end do...while
```

对二维数组deck进行初始化

随机选取某张未被选中的牌

57

```
51
52      // place card number in chosen slot of deck
53      deck[ row ][ column ] = card;
54   } // end for
55} // end function shuffle
56
57// deal cards in deck
58void DeckOfCards::deal()
59{
60   // initialize suit array
61   static const char *suit[ 4 ] =
62      { "Hearts", "Diamonds", "Clubs", "Spades" };
63
64   // initialize face array
65   static const char *face[ 13 ] =
66      { "Ace", "Deuce", "Three", "Four", "Five", "Six", "Seven",
67      "Eight", "Nine", "Ten", "Jack", "Queen", "King" };
```

58

```
68
69    // for each of the 52 cards
70    for ( int card = 1; card <= 52; card++ )
71    {
72       // loop through rows of deck
73       for ( int row = 0; row <= 3; row++ )
74       {
75          // loop through columns of deck for current row
76          for ( int column = 0; column <= 12; column++ )
77          {
78             // if slot contains current card, display card
79             if ( deck[ row ][ column ] == card )
80             {
81                cout << setw( 5 ) << right << face[ column ]
82                   << " of " << setw( 8 ) << left << suit[ row ]
83                   << ( card % 2 == 0 ? '\n' : '\t' );
84             } // end if
85          } // end innermost for
86       } // end inner for
87    } // end outer for
88} // end function deal
```

59

```
1 // Fig. 8.27: fig08_27.cpp
2 // Card shuffling and dealing program.
3 #include "DeckOfCards.h" // DeckOfCards class definition
4
5 int main()
6 {
7    DeckOfCards deckOfCards; // create DeckOfCards object
8
9    deckOfCards.shuffle(); // shuffle the cards in the deck
10   deckOfCards.deal(); // deal the cards in the deck
11   return 0; // indicates successful termination
12} // end main
```

60

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

61

程序运行效率的思考

1. function DeckofCards::shuffle()
2. function DeckofCards::deal()

是否可以改进？如何改进？

62

8.12 Function Pointers

- Pointers to functions 函数指针
  - Contain addresses of functions
    - Similar to how array name is address of first element
    - Function name is starting address of code that defines function 函数代码开始的地址
- Function pointers can be
  - Passed to functions
  - Returned from functions
  - Stored in arrays
  - Assigned to other function pointers

63

8.12 Function Pointers (Cont.)

返回值

函数形参列表

- Declare a function pointer
  - `bool ( *fPtr ) ( int, int );`
    `fPtr = function_name;`
- Calling functions using pointers
  - Assume function header parameter:

```
void func(int a, int b,
          bool ( *fPtr ) ( int, int ));
```

  - Execute function from pointer with either
    - `( * fPtr ) ( int1, int2 )`
    - `fPtr( int1, int2 )`

64

```
1 // Fig. 8.28: fig08_28.cpp
2 // Multipurpose sorting program using function pointers.
10
11// prototypes
12void selectionSort( int [], const int, bool (*)( int, int ) );
13void swap( int * const, int * const );
14bool ascending( int, int ); // implements ascending order
15bool descending( int, int ); // implements descending order
16
17int main()
18{
19   const int arraySize = 10;
20   int order; // 1 = ascending, 2 = descending
21   int counter; // array index
22   int a[ arraySize ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
23
24   cout << "Enter 1 to sort in ascending order,\n"
25      << "Enter 2 to sort in descending order: ";
26   cin >> order;
27   cout << "\nData items in original order\n";
```

65

```
28
29   // output original array
30   for ( counter = 0; counter < arraySize; counter++ )
31      cout << setw( 4 ) << a[ counter ];
32
33   // sort array in ascending order; pass function ascending
34   // as an argument to specify ascending sorting order
35   if ( order == 1 )
36   {
37      selectionSort( a, arraySize, ascending );
38      cout << "\nData items in ascending order\n";
39   } // end if
40
41   // sort array in descending order; pass function descending
42   // as an argument to specify descending sorting order
43   else
44   {
45      selectionSort( a, arraySize, descending );
46      cout << "\nData items in descending order\n";
47   } // end else part of if...else
48
49   // output sorted array
50   for ( counter = 0; counter < arraySize; counter++ )
51      cout << setw( 4 ) << a[ counter ];
52
53   cout << endl;
54   return 0; // indicates successful termination
55} // end main
```

66

```
57// multipurpose selection sort; the parameter compare is a pointer to
58// the comparison function that determines the sorting order
59void selectionSort( int work[], const int size,
60                    bool (*compare)( int, int ) )
61{
62   int smallestOrLargest; // index of smallest (or largest) element
64   // loop over size - 1 elements
65   for ( int i = 0; i < size - 1; i++ )
66   {
67      smallestOrLargest = i; // first index of remaining vector
68
69      // loop to find index of smallest (or largest) element
70      for ( int index = i + 1; index < size; index++ )
71         if ( !(*compare)( work[ smallestOrLargest ], work[ index ] ) )
72            smallestOrLargest = index;
73
74      swap( &work[ smallestOrLargest ], &work[ i ] );
75   } // end if
76} // end function selectionSort
```

67

```
87// determine whether element a is less than
88// element b for an ascending order sort
89bool ascending( int a, int b )
90{
91   return a < b; // returns true if a is less than b
92} // end function ascending
93
94// determine whether element a is greater than
95// element b for a descending order sort
96bool descending( int a, int b )
97{
98   return a > b; // returns true if a is greater than b
99} // 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
```

68

8.12 Function Pointers (Cont.)

- Arrays of pointers to functions 函数指针数组
  - Menu-driven systems 菜单式系统
    - Pointers to each function stored in array of pointers to functions
      - All functions must have same return type and same parameter types 所有函数必须有相同的返回值类型和相同的形参列表
    - Menu choice determines subscript into array of function pointers

69

```
1 // Fig. 8.29: fig08_29.cpp
2 // Demonstrating an array of pointers to functions.
3 #include <iostream>
4 using std::cout;
5 using std::cin;
6 using std::endl;
7
8 // function prototypes -- each function performs similar actions
9 void function0( int );
10void function1( int );
11void function2( int );
12
13int main()
14{
15   // initialize array of 3 pointers to functions that each
16   // take an int argument and return void
17   void (*f[ 3 ])( int ) = { function0, function1, function2 };
18
19   int choice;
20
21   cout << "Enter a number between 0 and 2, 3 to end: ";
22   cin >> choice;
```

70

```
23
24    // process user's choice
25    while ( ( choice >= 0 ) && ( choice < 3 ) )
26    {
27       // invoke the function at location choice in
28       // the array f and pass choice as an argument
29       (*f[ choice ])( choice );
30
31       cout << "Enter a number between 0 and 2, 3 to end: ";
32       cin >> choice;
33    } // end while
34
35    cout << "Program execution completed." << endl;
36    return 0; // indicates successful termination
37 } // end main
38
39 void function0( int a )
40 {
41    cout << "You entered " << a << " so function0 was called\n\n";
42 } // end function function0
43
44 void function1( int b )
45 {
46    cout << "You entered " << b << " so function1 was called\n\n";
47 } // end function function1
```

71

```
48
49 void function2( int c )
50 {
51    cout << "You entered " << c << " so function2 was called\n\n";
52 } // 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.
```

72

8.13 Introduction to Pointer-Based String Processing

- C++风格的字符串
  - Chapter 3, GradeBook中定义的string courseName; (P87)
  - courseName.length();
  - courseName.substr(0, 25);
- C风格的字符串（以空字符null结束的字符数组）
  - char ca2[]={'c','+','+','\0'}; //explict null
    char ca3[]="C++"; //null terminator added automatically
    const char *cp="C++"; //null terminator added automatically
    char *cp2=ca2; //points to first element of a null-terminated char array

8.13.1 Fundamentals of Characters and Pointer-Based Strings (Cont.)

- 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'`)

8.13.1 Fundamentals of Characters and Pointer-Based Strings (Cont.)

- `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' );
```

75

8.13.2 String Manipulation Functions of the String-Handling Library

- 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>`

76

Function prototype	Function description
`char *strcpy( char *s1, const char *s2 );`	Copies the string `s2` into the character array `s1`. The value of `s1` is returned.
`char *strncpy( char *s1, const char *s2, size_t n );`	Copies at most `n` characters of the string `s2` into the character array `s1`. The value of `s1` is returned.
`char *strcat( char *s1, const char *s2 );`	Appends the string `s2` to `s1`. The first character of `s2` overwrites the terminating null character of `s1`. The value of `s1` is returned.
`char *strncat( char *s1, const char *s2, size_t n );`	Appends at most `n` characters of string `s2` to string `s1`. The first character of `s2` overwrites the terminating null character of `s1`. The value of `s1` is returned.
`int strcmp( const char *s1, const char *s2 );`	Compares the string `s1` with the string `s2`. The function returns a value of zero, less than zero (usually `-1`) or greater than zero (usually `1`) if `s1` is equal to, less than or greater than `s2`, respectively.

77

Function prototype	Function description
`int strncmp( const char *s1, const char *s2, size_t n );`	Compares up to `n` characters of the string `s1` with the string `s2`. The function returns zero, less than zero or greater than zero if the `n`-character portion of `s1` is equal to, less than or greater than the corresponding `n`-character portion of `s2`, respectively.
`char *strtok( char *s1, const char *s2 );`	A sequence of calls to `strtok` breaks string `s1` into "tokens"—logical pieces such as words in a line of text. The string is broken up based on the characters contained in string `s2`. For instance, if we were to break the string `"this:is:a:string"` into tokens based on the character `':'`, the resulting tokens would be `"this"`, `"is"`, `"a"` and `"string"`. Function `strtok` returns only one token at a time, however. The first call contains `s1` as the first argument, and subsequent calls to continue tokenizing the same string contain `NULL` as the first argument. A pointer to the current token is returned by each call. If there are no more tokens when the function is called, `NULL` is returned.
`size_t strlen( const char *s );`	Determines the length of string `s`. The number of characters preceding the terminating null character is returned.

78

8.13.2 String Manipulation Functions of the String-Handling Library (Cont.)

- Copying strings
  - `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
      - 当n小于s2长度+1时，需要手工添加‘\0’

79

Common Programming Error 8.20

When using `strncpy`, the terminating null character of the second argument (a `char *` string) will not be copied if the number of characters specified by `strncpy`'s third argument is not greater than the second argument's length. In that case, a fatal error may occur if the programmer does not manually terminate the resulting `char *` string with a null character.

80

```
1 // Fig. 8.31: fig08_31.cpp
2 // Using strcpy and strncpy.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 #include <cstring> // prototypes for strcpy and strncpy
8 using std::strcpy;
9 using std::strncpy;
10
11 int main()
12 {
13    char x[] = "Happy Birthday to You"; // string length 21
14    char y[ 25 ];
15    char z[ 15 ];
16
17    strcpy( y, x ); // copy contents of x into y
18
19    cout << "The string in array x is: " << x
20       << "\nThe string in array y is: " << y << '\n';
```



```
21
22    // copy first 14 characters of x into z
23    strncpy( z, x, 14 ); // does not copy null character
24    z[ 14 ] = '\0'; // append '\0' to z's contents
25
26    cout << "The string in array z is: " << z << endl;
27    return 0; // indicates successful termination
28 } // 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
```

8.13.2 String Manipulation Functions of the String-Handling Library (Cont.)

- Concatenating strings
  - `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


```
1 // Fig. 8.32: fig08_32.cpp
2 // Using strcat and strncat.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 #include <cstring> // prototypes for strcat and strncat
8 using std::strcat;
9 using std::strncat;
10
11int main()
12{
13   char s1[ 20 ] = "Happy "; // length 6
14   char s2[] = "New Year "; // length 9
15   char s3[ 40 ] = "";
16
17   cout << "s1 = " << s1 << "\ns2 = " << s2;
18
19   strcat( s1, s2 ); // concatenate s2 to s1 (length 15)
20
21   cout << "\n\nAfter strcat(s1, s2):\ns1 = " << s1 << "\ns2 = " << s2;
22
23   // concatenate first 6 characters of s1 to s3
24   strncat( s3, s1, 6 ); // places '\0' after last character
25
26   cout << "\n\nAfter strncat(s3, s1, 6):\ns1 = " << s1
27      << "\ns3 = " << s3;
```

```
28
29    strcat( s3, s1 ); // concatenate s1 to s3
30    cout << "\n\nAfter strcat(s3, s1):\ns1 = " << s1
31       << "\ns3 = " << s3 << endl;
32    return 0; // indicates successful termination
33 } // end main
```

```
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
```

85

8.13.2 String Manipulation Functions of the String-Handling Library (Cont.)

- Comparing strings
  - `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

86

```
1 // Fig. 8.33: fig08_33.cpp
2 // Using strcmp and strncmp.
9
10#include <cstring> // prototypes for strcmp and strncmp
11using std::strcmp;
12using std::strncmp;
13
14int main()
15{
16   char *s1 = "Happy New Year";
17   char *s2 = "Happy New Year";
18   char *s3 = "Happy Holidays";
19
20   cout << "s1 = " << s1 << "\ns2 = " << s2 << "\ns3 = " << s3
21      << "\n\nstrcmp(s1, s2) = " << setw( 2 ) << strcmp( s1, s2 )
22      << "\nstrcmp(s1, s3) = " << setw( 2 ) << strcmp( s1, s3 )
23      << "\nstrcmp(s3, s1) = " << setw( 2 ) << strcmp( s3, s1 );
24
25   cout << "\n\nstrncmp(s1, s3, 6) = " << setw( 2 )
26      << strncmp( s1, s3, 6 ) << "\nstrncmp(s1, s3, 7) = " << setw( 2 )
27      << strncmp( s1, s3, 7 ) << "\nstrncmp(s3, s1, 7) = " << setw( 2 )
28      << strncmp( s3, s1, 7 ) << endl;
29   return 0; // indicates successful termination
30} // end main
```


```
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
```

78

72

8.13.2 String Manipulation Functions of the String-Handling Library (Cont.)

- Tokenizing 记号化字符串
  - Breaking strings into tokens（记号）
    - 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)

89

8.13.2 String Manipulation Functions of the String-Handling Library (Cont.)

- 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（ ‘\0’）
      - Subsequent calls continue tokenizing
        - » Call with first argument NULL
        - » Stores pointer to remaining string in a `static` variable
    - Returns pointer to current token

90

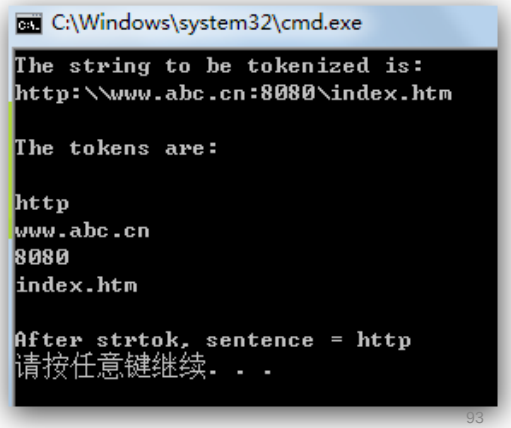
```cpp
// Fig. 8.34: fig08_34.cpp
// Using strtok.
#include <iostream>

#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
```

91

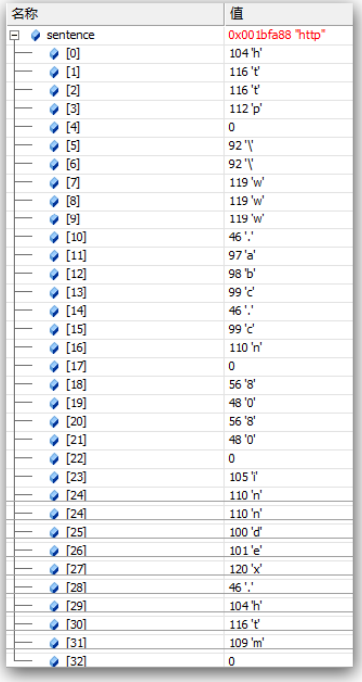
```
The string to be tokenized is:
This is a sentence with 7 tokens

The tokens are:

This
is
a
sentence
with
7
tokens

After strtok, sentence = This
```

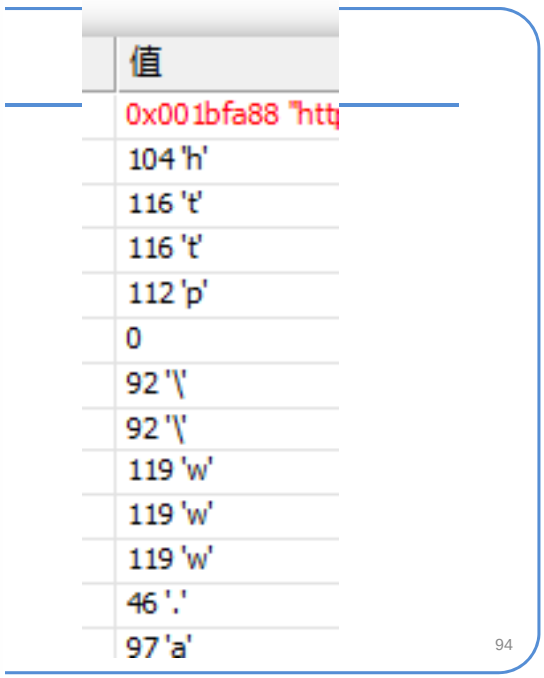
92

```
int main()
{
  char sentence[] = "http:\\\\www.abc.cn:8080\\index.htm";
  char *tokenPtr;
  cout << "The string to be tokenized is:\n" << sentence
    << "\n\nThe tokens are:\n\n";
  // begin tokenization of se
  tokenPtr = strtok( sentenc
  while ( tokenPtr != NULL
  {
    cout << tokenPtr << '\n'
         tokenPtr = strtok( N
  } // end while
  cout << "\nAfter strtok, s
  return 0; // indicates succ
} // end main
```



93





94

A sequence of calls to this function split *str* into tokens, which are sequences of contiguous characters separated by any of the characters that are part of *delimiters*.

To determine the beginning and the end of a token, the function first scans from the starting location for the first character not contained in *delimiters* (which becomes the *beginning of the token*). And then scans starting from this *beginning of the token* for the first character contained in *delimiters*, which becomes the *end of the token*. This *end of the token* is automatically replaced by a null-character by the function, and the *beginning of the token* is returned by the function.

8.13.2 String Manipulation Functions of the String-Handling Library (Cont.)

- 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

96

```
1 // Fig. 8.35: fig08_35.cpp
2 // Using strlen.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 #include <cstring> // prototype for strlen
8 using std::strlen;
9
10int main()
11{
12   char *string1 = "abcdefghijklmnopqrstuvwxyz";
13   char *string2 = "four";
14   char *string3 = "Boston";
15
16   cout << "The length of \"" << string1 << "\" is " << strlen( string1 )
17      << "\nThe length of \"" << string2 << "\" is " << strlen( string2 )
18      << "\nThe length of \"" << string3 << "\" is " << strlen( string3 )
19      << endl;
20   return 0; // indicates successful termination
21} // end main
```

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

97

10.6 Dynamic Memory Management with Operators new and delete

- Dynamic memory management
  - Enables programmers to allocate and deallocate memory for any built-in or user-defined type
  - Performed by operators new and delete
  - For example, dynamically allocating memory for an array instead of using a fixed-size array

98

10.6 Dynamic Memory Management with Operators `new` and `delete` (Cont.)

- Operator `new`
  - Allocates (i.e., reserves) storage of the proper size for an object at execution time
  - Calls a constructor to initialize the object
  - Returns a pointer of the type specified to the right of `new`
  - Can be used to dynamically allocate any fundamental type (such as `int` or `double`) or any class type
- Free store
  - Sometimes called the heap
  - Region of memory assigned to each program for storing objects created at execution time

10.6 Dynamic Memory Management with Operators `new` and `delete` (Cont.)

- Operator `delete`
  - Destroys a dynamically allocated object
  - Calls the destructor for the object
  - Deallocates (i.e., releases) memory from the free store
  - The memory can then be reused by the system to allocate other objects

10.6 Dynamic Memory Management with Operators new and delete (Cont.)

- Initializing an object allocated by new
  - Initializer for a newly created fundamental-type variable
    - Example
      - `double *ptr = new double( 3.14159 );`
  - Specify a comma-separated list of arguments to the constructor of an object
    - Example
      - `Time *timePtr = new Time( 12, 45, 0 );`

101

10.6 Dynamic Memory Management with Operators new and delete (Cont.)

- new operator can be used to allocate arrays dynamically
  - Dynamically allocate a 10-element integer array:
    `int *gradesArray = new int[ 10 ];`
  - Size of a dynamically allocated array
    - Specified using any integral expression that can be evaluated at execution time

102

10.6 Dynamic Memory Management with Operators new and delete (Cont.)

- Delete a dynamically allocated array:

```
delete [] gradesArray;
```

  - This deallocates the array to which `gradesArray` points
  - If the pointer points to an array of objects
    - First calls the destructor for every object in the array
    - Then deallocates the memory
  - If the statement did not include the square brackets (`[]`) and `gradesArray` pointed to an array of objects
    - Only the first object in the array would have a destructor call

103

练习

- 各编写一条C++语句完成指定任务。假设已经声明了：

```
double num1 = 7.3, num2;
```

  - 声明变量fPtr指向double类型对象的指针；
  - 把变量num1的地址赋给指针变量fPtr;
  - 打印fPtr所指向对象的值；
  - 把fPtr所指对象的值赋给变量num2;
  - 打印num1的地址；
  - 打印存储在fPtr中的地址。打印的地址是否和num1的地址一样？

104