Chapter 8
Pointers and Pointer-Based Strings

OBJECTIVES

- **What pointers are.**
- **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 declare and use arrays of C-style strings.**

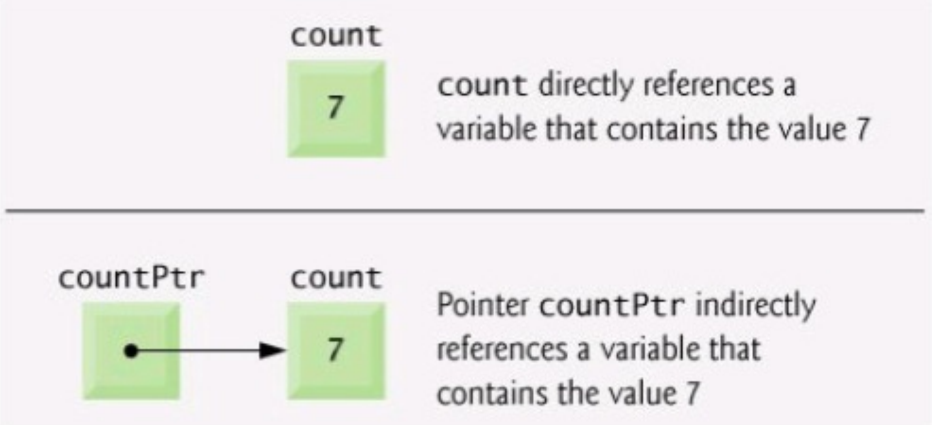
Topics

- **8.1 Pointer and Pointer Parameter**
- **8.2 Using const with Pointers**
- **8.3 sizeof Operators**
- **8.4 Selection Sort Using Pass-by-Reference**
- 8.5 Pointer Expressions and Pointer Arithmetic
- 8.6 Relationship Between Pointers and Arrays
- 8.7 Introduction to Pointer-Based String Processing
- 8.8 Arrays of Pointers

8.1 Pointer and Pointer Parameter

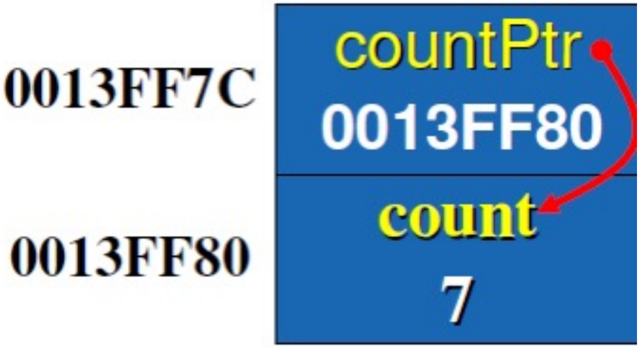
(1) **Pointer Variable(指针变量): Variables contain memory addresses as their values.**

```
int *countPtr, count = 7;
countPtr = &count; // 取地址符
```



count directly references a variable that contains the value 7

Pointer countPtr indirectly references a variable that contains the value 7



0013FF7C

countPtr

0013FF80

0013FF80

count

7

8.1 Pointer and Pointer Parameter

(2) 声明两个指针变量:

□ double *ptrX, ptrY;

// Error, 仅声明了一个指针变量

□ double *ptrX, *ptrY;

指针变量的初始值:

① int *countPtr = 0;

② int *countPtr = NULL;

③ int *countPtr;

countPtr = 0; // NULL;

8.1 Pointer and Pointer Parameter

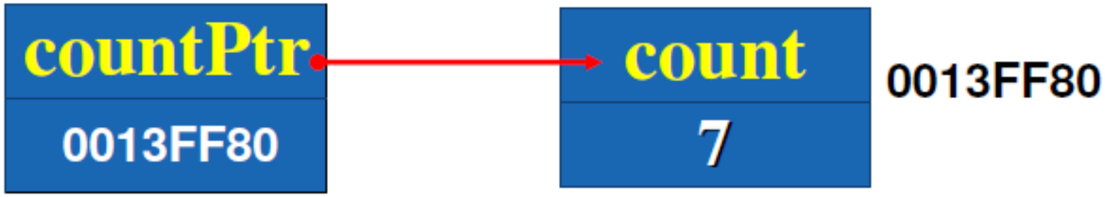
(3) Address operator (&): 单目操作符, 返回操作数的内存地址

int *countPtr, count = 7;

countPtr = &count; // 取地址符

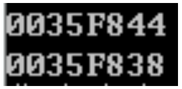
❑ int &countRef = count; // 引用符

always preceded by a data-type name



8.1 Pointer and Pointer Parameter

- 1. int a = 10, b = 20;
- 2. cout << &a << endl;
- 3. cout << &b << endl;
- 4. cout << &(a + b) << endl;



```
0035F844
0035F838
```

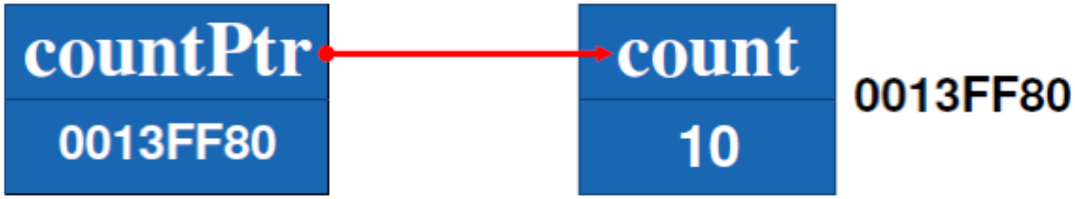
8.1 Pointer and Pointer Parameter

(4) * operator

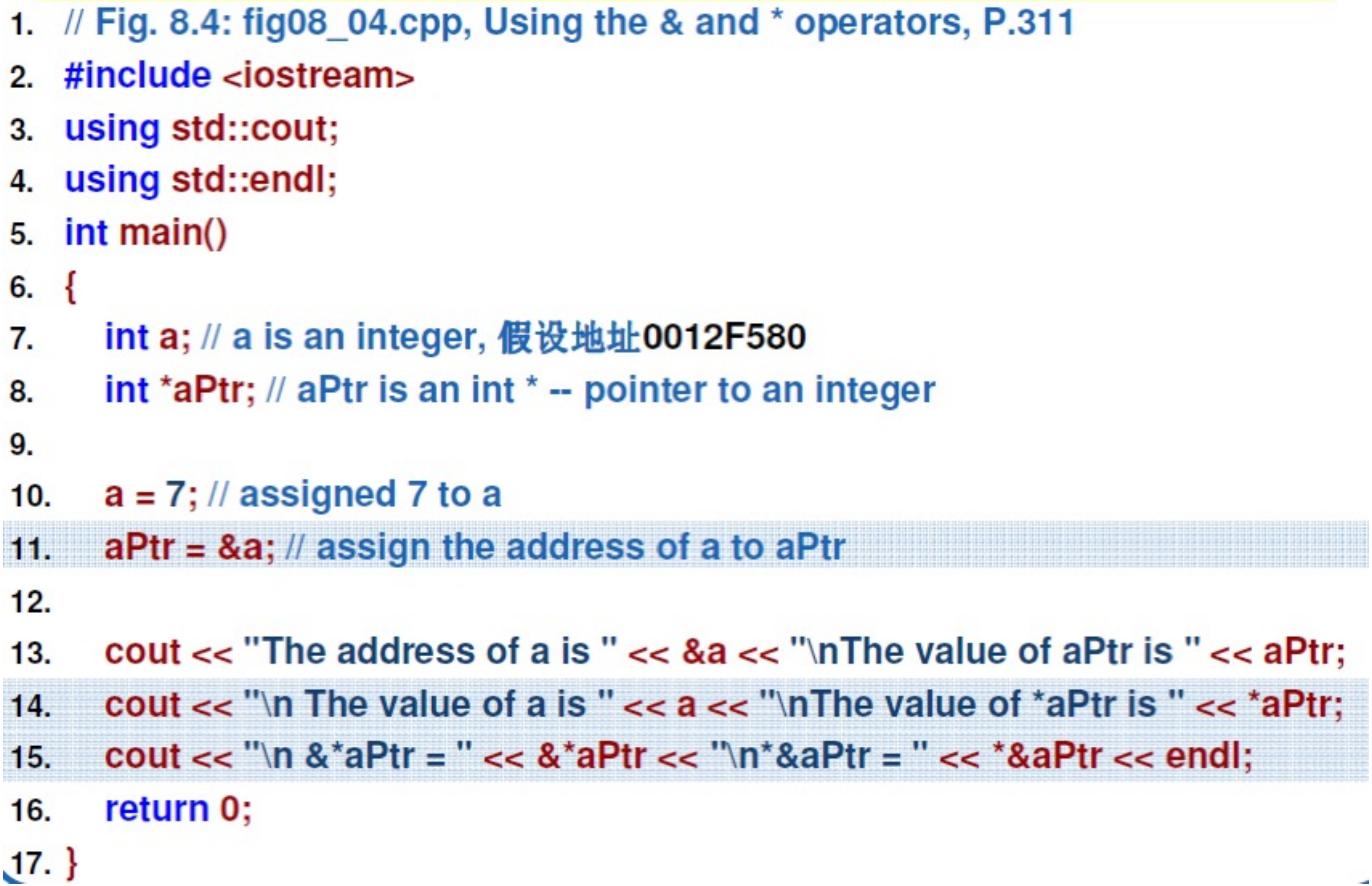
- ❑ indirection/dereferencing operator(间接/解引用), 返回指针指向变量的别名
- ❑ int *countPtr, count = 7;

  countPtr = &count;

  *countPtr = 10; // *countPtr是count的别名



8.1 Pointer and Pointer Parameter

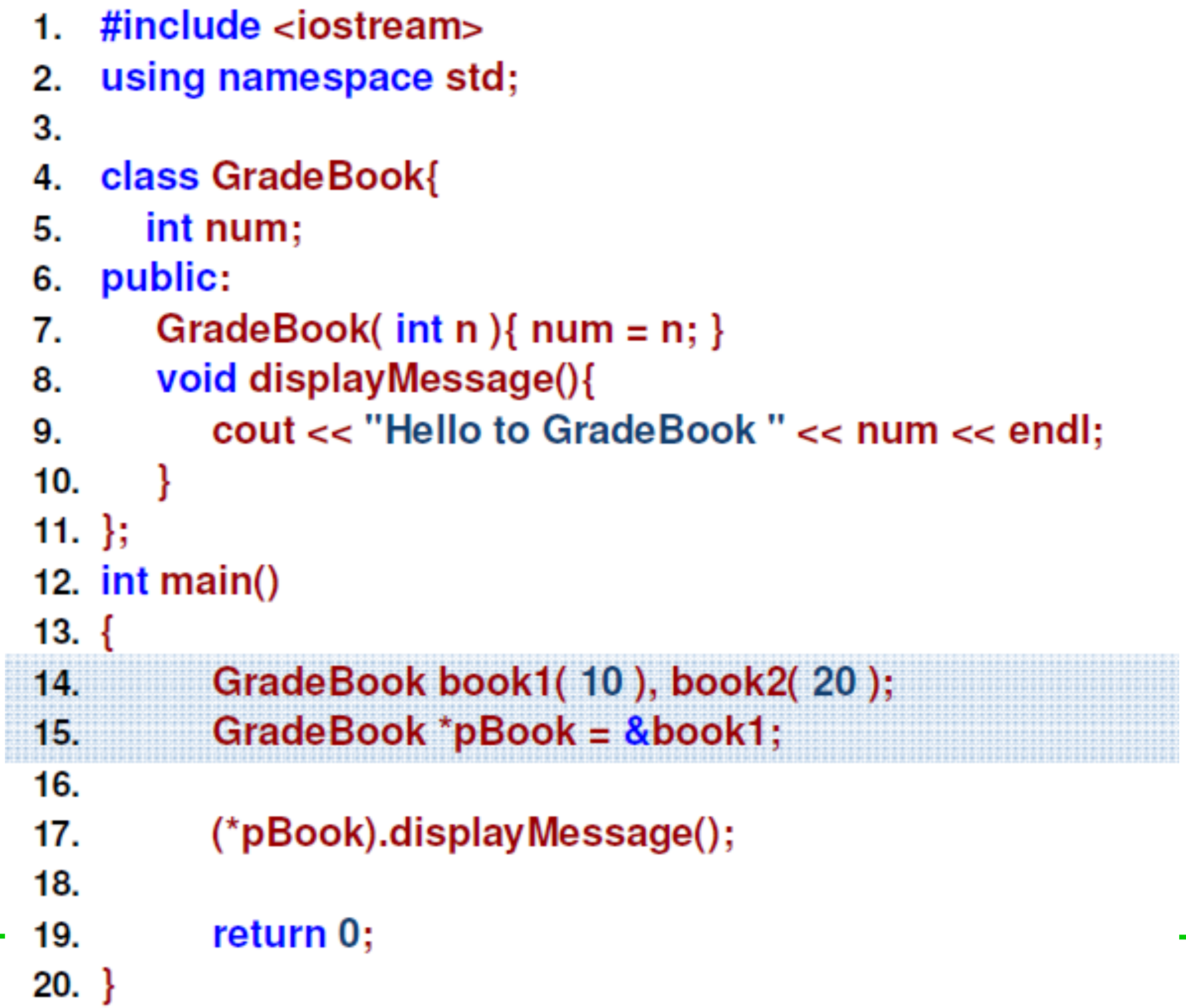


```
// Fig. 8.4: fig08_04.cpp, Using the & and * operators, P.311
#include <iostream>
using std::cout;
using std::endl;
int main()
{
    int a; // a is an integer, 假设地址0012F580
    int *aPtr; // aPtr is an int * -- pointer to an integer

    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 The value of a is " << a << "\nThe value of *aPtr is " << *aPtr;
    cout << "\n &*aPtr = " << &*aPtr << "\n*&aPtr = " << *&aPtr << endl;
    return 0;
}
```

8.1 Pointer and Pointer Parameter



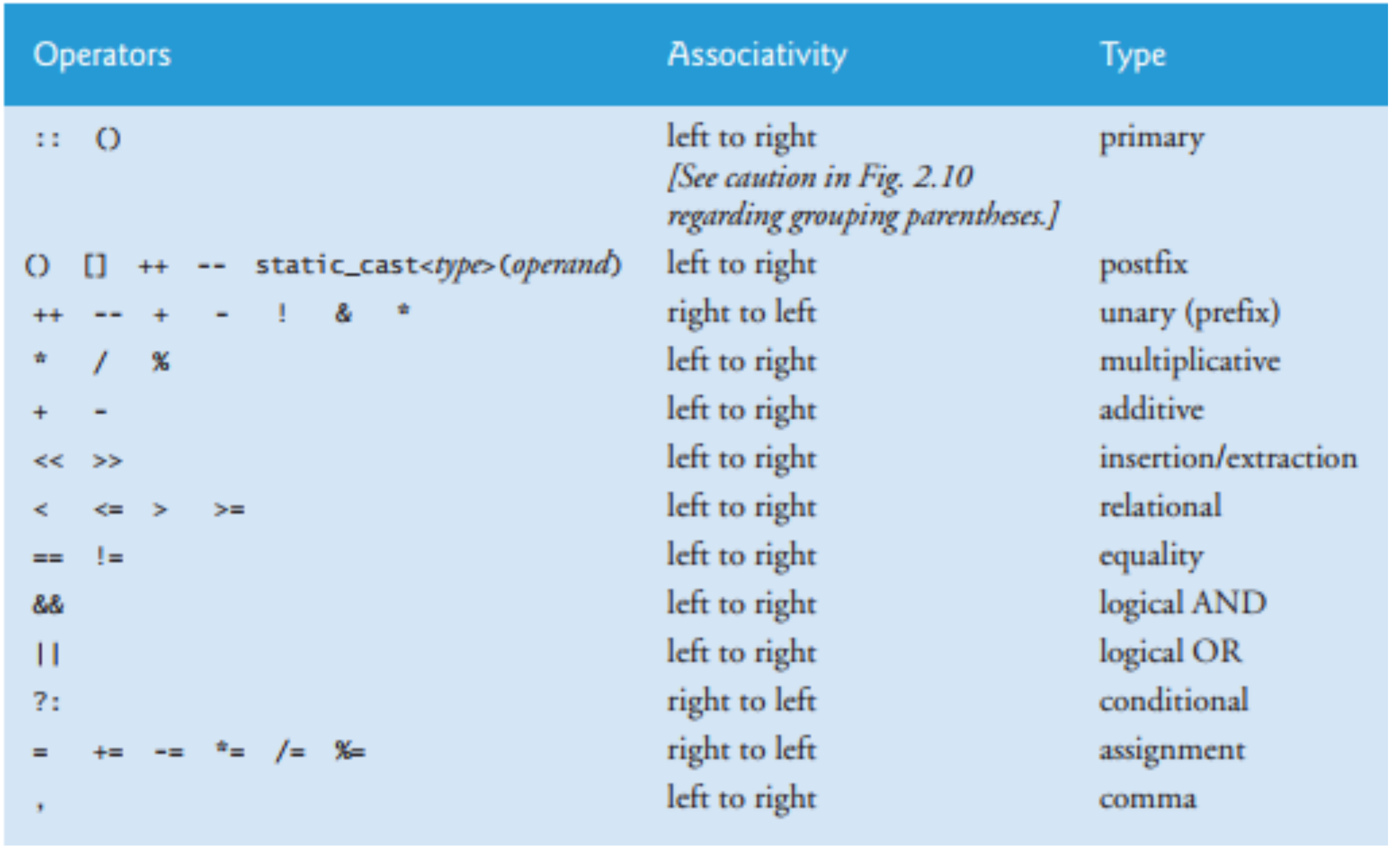
```
#include <iostream>
using namespace std;

class GradeBook{
    int num;
public:
     GradeBook( int n ){ num = n; }
     void displayMessage(){
          cout << "Hello to GradeBook " << num << endl;
     }
};
int main()
{
          GradeBook book1( 10 ), book2( 20 );
          GradeBook *pBook = &book1;

          (*pBook).displayMessage();

          return 0;
}
```

8.1 Pointer and Pointer Parameter



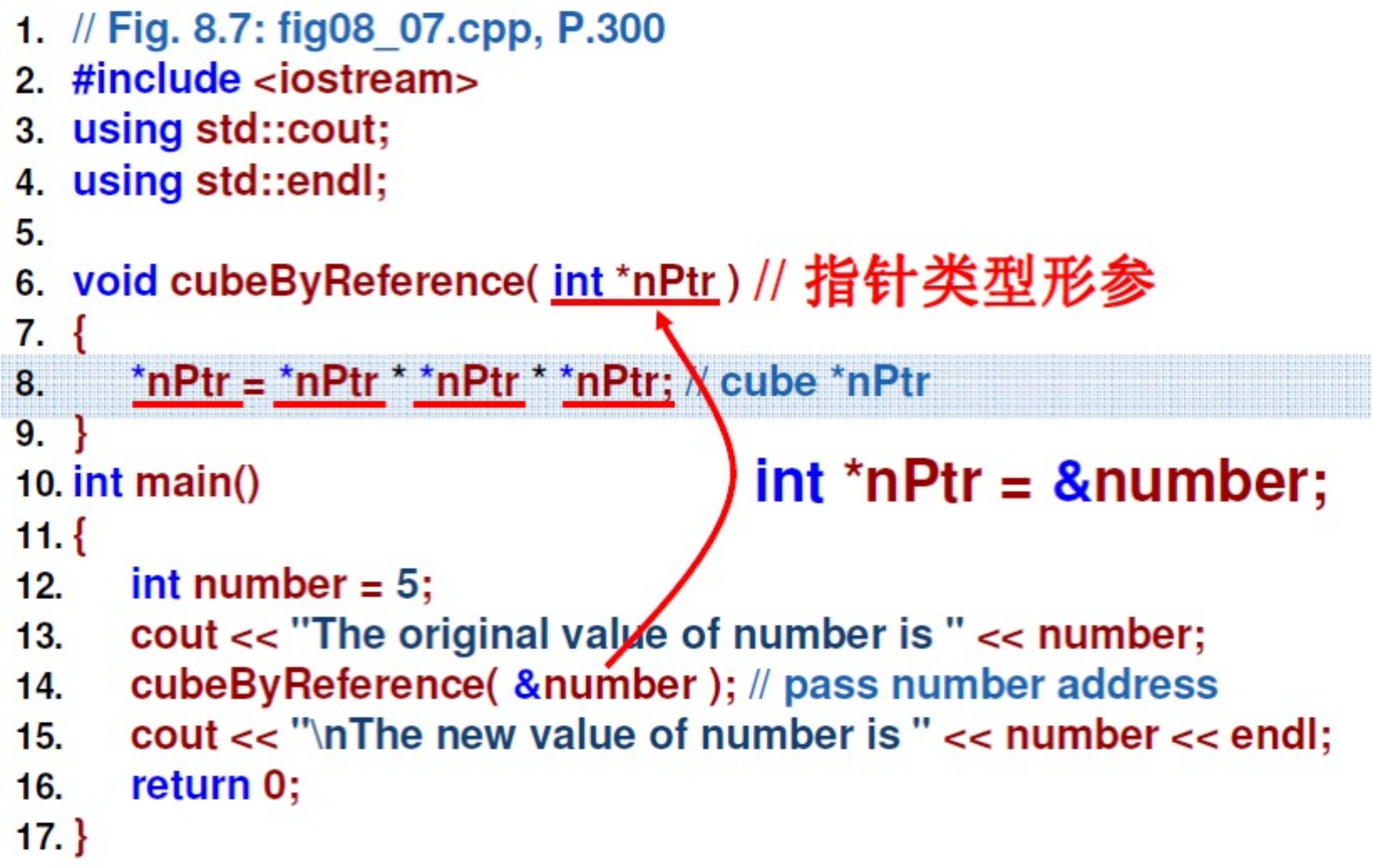
Operators	Associativity	Type
`::` `()`	left to right *[See caution in Fig. 2.10 regarding grouping parentheses.]*	primary
`()` `[]` `++` `--` `static_cast<`*type*`>(`*operand*`)`	left to right	postfix
`++` `--` `+` `-` `!` `&` `*`	right to left	unary (prefix)
`*` `/` `%`	left to right	multiplicative
`+` `-`	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

8.1 Pointer and Pointer Parameter

(5) 函数参数传递的两种方式:

- **Pass-by-Value, 传值**
- **Pass-by-Reference, 传引用**
  - **Reference Parameter, 引用参数**
  - **Pointer Parameter, 指针参数**

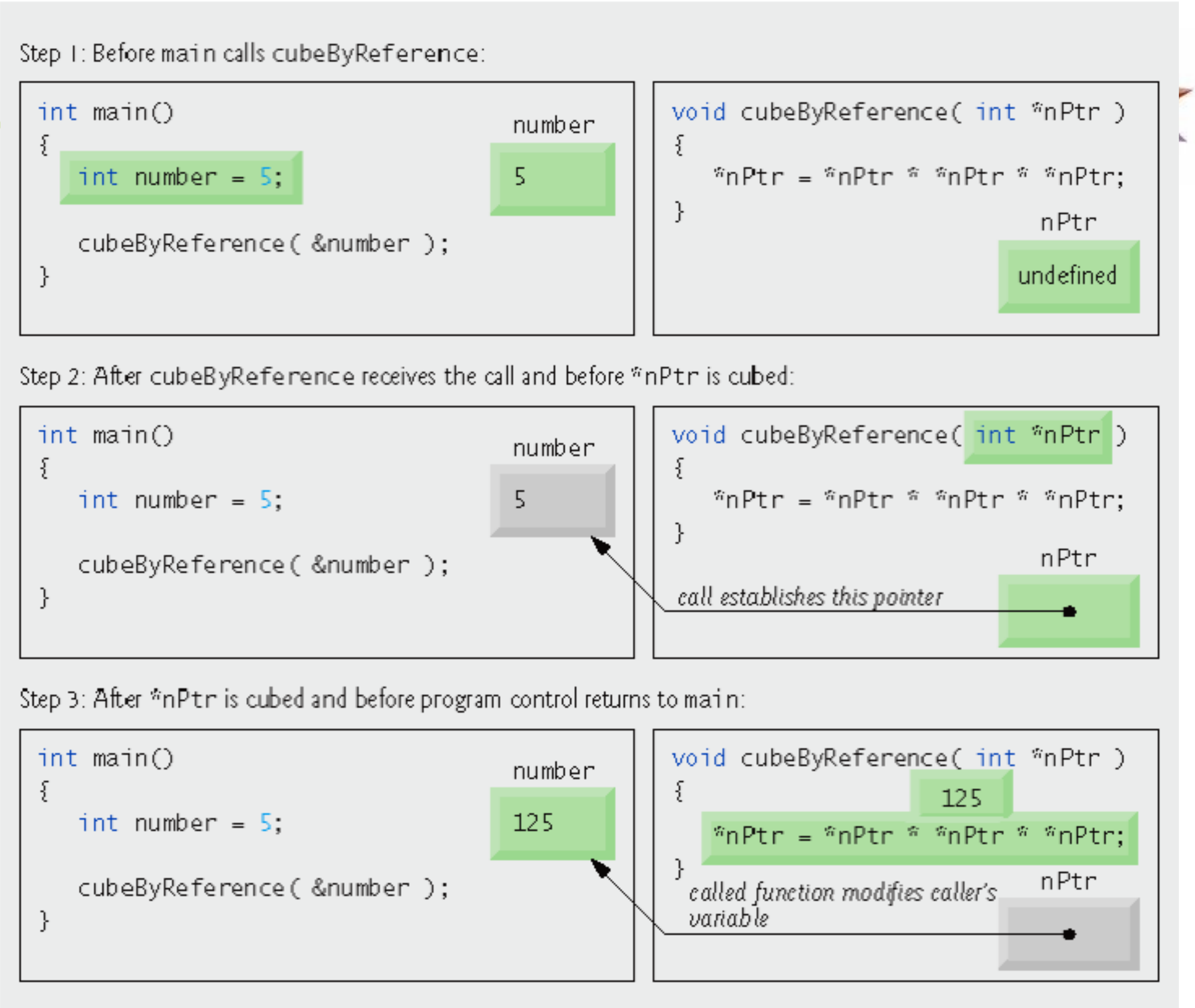
8.1 Pointer and Pointer Parameter



```
// Fig. 8.7: fig08_07.cpp, P.300
#include <iostream>
using std::cout;
using std::endl;

void cubeByReference( int *nPtr ) // 指针类型形参
{
    *nPtr = *nPtr * *nPtr * *nPtr; // cube *nPtr
}
int main()
{
    int number = 5;
    cout << "The original value of number is " << number;
    cubeByReference( &number ); // pass number address
    cout << "\nThe new value of number is " << number << endl;
    return 0;
}
```

int *nPtr = &number;



Step 1: Before main calls cubeByReference:

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

   cubeByReference( &number );
}
```

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

Step 2: After cubeByReference receives the call and before *nPtr is cubed:

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

   cubeByReference( &number );
}
```

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

call establishes this pointer

Step 3: After *nPtr is cubed and before program control returns to main:

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

   cubeByReference( &number );
}
```

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

called function modifies caller's variable

Topics

- □ 8.1 Pointer and Pointer Parameter
- □ **8.2 Using const with Pointers**
- □ 8.3 sizeof Operators
- □ 8.4 Selection Sort Using Pass-by-Reference
- □ 8.5 Pointer Expressions and Pointer Arithmetic
- □ 8.6 Relationship Between Pointers and Arrays
- □ 8.7 Introduction to Pointer-Based String Processing
- □ 8.8 Arrays of Pointers

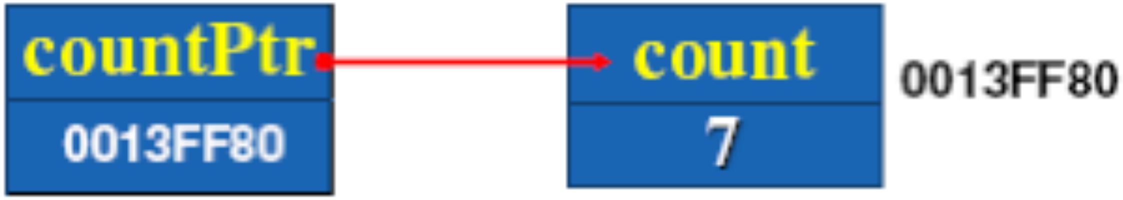
8.2 Using const with Pointers

- 常量变量

  const int m = 1000; //声明时必须初始化

  表示int型变量m为常量变量
- 问题: 指针类型的变量p如何声明为常量?

  const int * p = ……
- Constant Pointer: 指针常量, 指针类型的常量
- Pointer to Constant: 常量指针, 指向常量的指针



8.2 Using const with Pointers

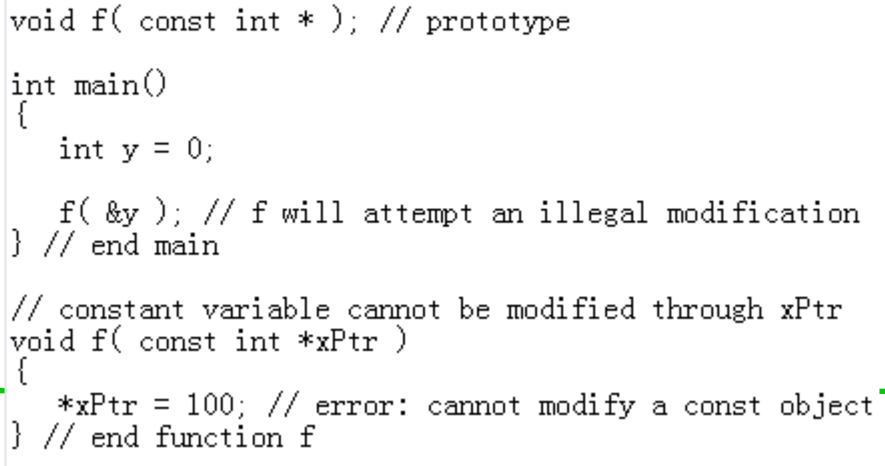
分类	non-constant data	constant data
non-constant pointer	int * p1	const int * p2
constant pointer	int * const p3	const int * const p4

- □ **p1**: **Nonconstant pointer to Nonconstant data**
- □ 数据可以修改: **data can be modified through the dereferenced pointer**
- □ 指针可以修改: **and the pointer can be modified to point to other data**

8.2 Using const with Pointers

分　类	non-constant data	constant data
non-constant pointer	int * p1	const int * p2
constant pointer	int * const p3	const int * const p4

- ❑ **p2**: **Nonconstant pointer to Constant data**
- ❑ 常量指针,指向常量的指针
- ❑ 指针可以修改, 数据不能修改



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

int main()
{
   int y = 0;

   f( &y ); // f will attempt an illegal modification
} // end main

// constant variable cannot be modified through xPtr
void f( const int *xPtr )
{
   *xPtr = 100; // error: cannot modify a const object
} // end function f
```

8.2 Using const with Pointers

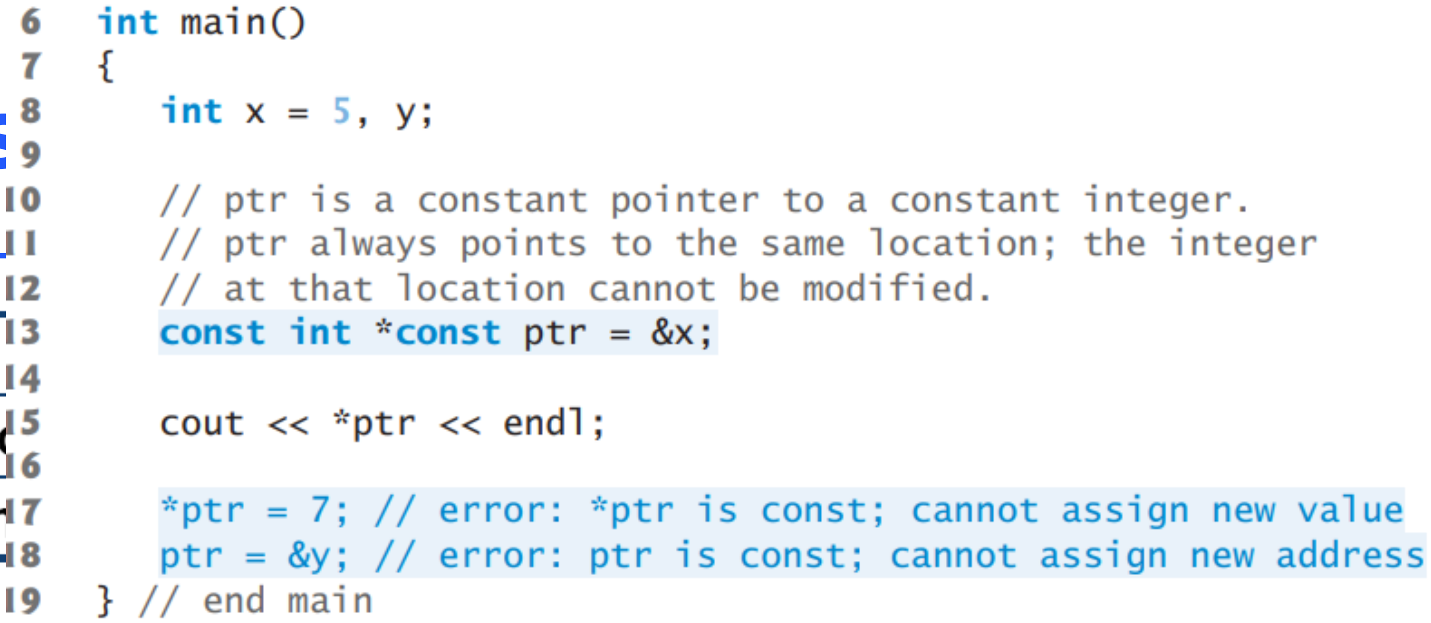
分　类	non-constant data	constant data
non-constant pointer	int * p1	const int * p2
constant pointer	int * const p3	const int * const p4

- p3: Constant pointer to Nonconstant data
- 指针常量,指针类型的常量
- 指针不能修改, 数据可以修改
- 注意: 声明时必须进行初始化, 或作为形参通过实参初始化

```
int m1 = 1000, m2 = 2000;
int * const p3 = &m1;   *p3 = 1500;
p3 = &m2;
```

8.2 Us

分 类
non-constant p
constant poi



```
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
} // end main
```

- p4: Constant pointer to Constant data
- 指向常量的指针常量,常量指针+指针常量
- 指针不能修改, 数据不能修改
- 声明时必须进行初始化, 或作为形参通过实参初始化

Topics

- □ 8.1 Pointer and Pointer Parameter
- □ 8.2 Using const with Pointers
- □ **8.3 sizeof Operators**
- □ 8.4 Selection Sort Using Pass-by-Reference
- □ 8.5 Pointer Expressions and Pointer Arithmetic
- □ 8.6 Relationship Between Pointers and Arrays
- □ 8.7 Introduction to Pointer-Based String Processing
- □ 8.8 Arrays of Pointers

8.3 sizeof Operators

- ❑ Compile-time operator: determine the size of operand (占用的以字节为单位的内存空间大小)
- ❑ Operand(操作数)

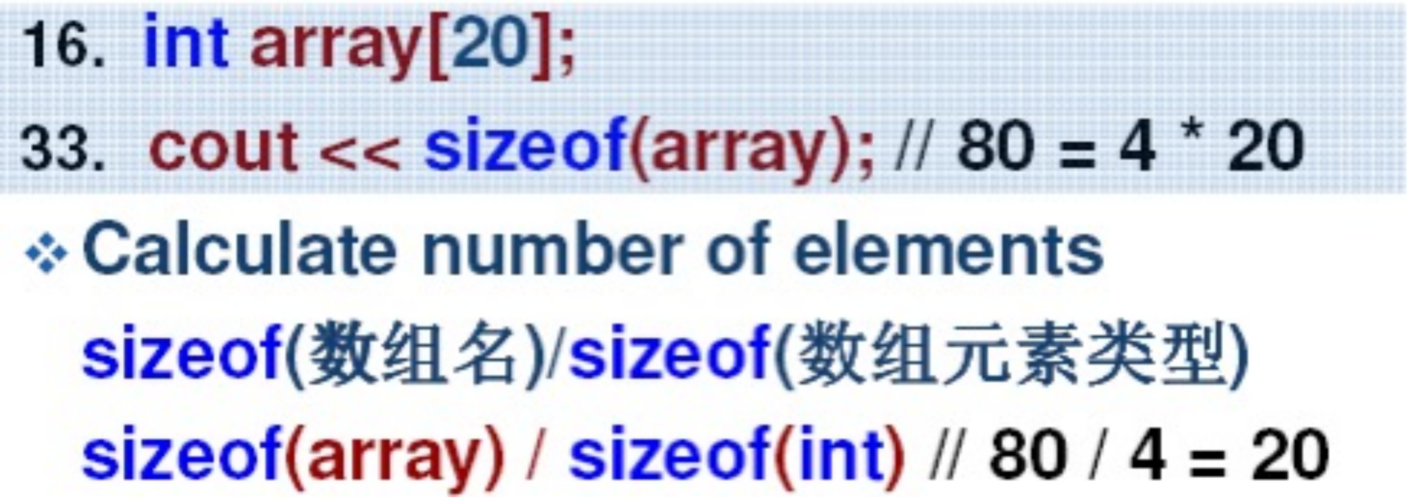
① 变量和常量名: 是否带括号可选

```
int number;
sizeof( number )
sizeof number
```

② 类型名: 必须带括号

```
sizeof( char )
sizeof( GradeBook )
```

8.3 sizeof Operators



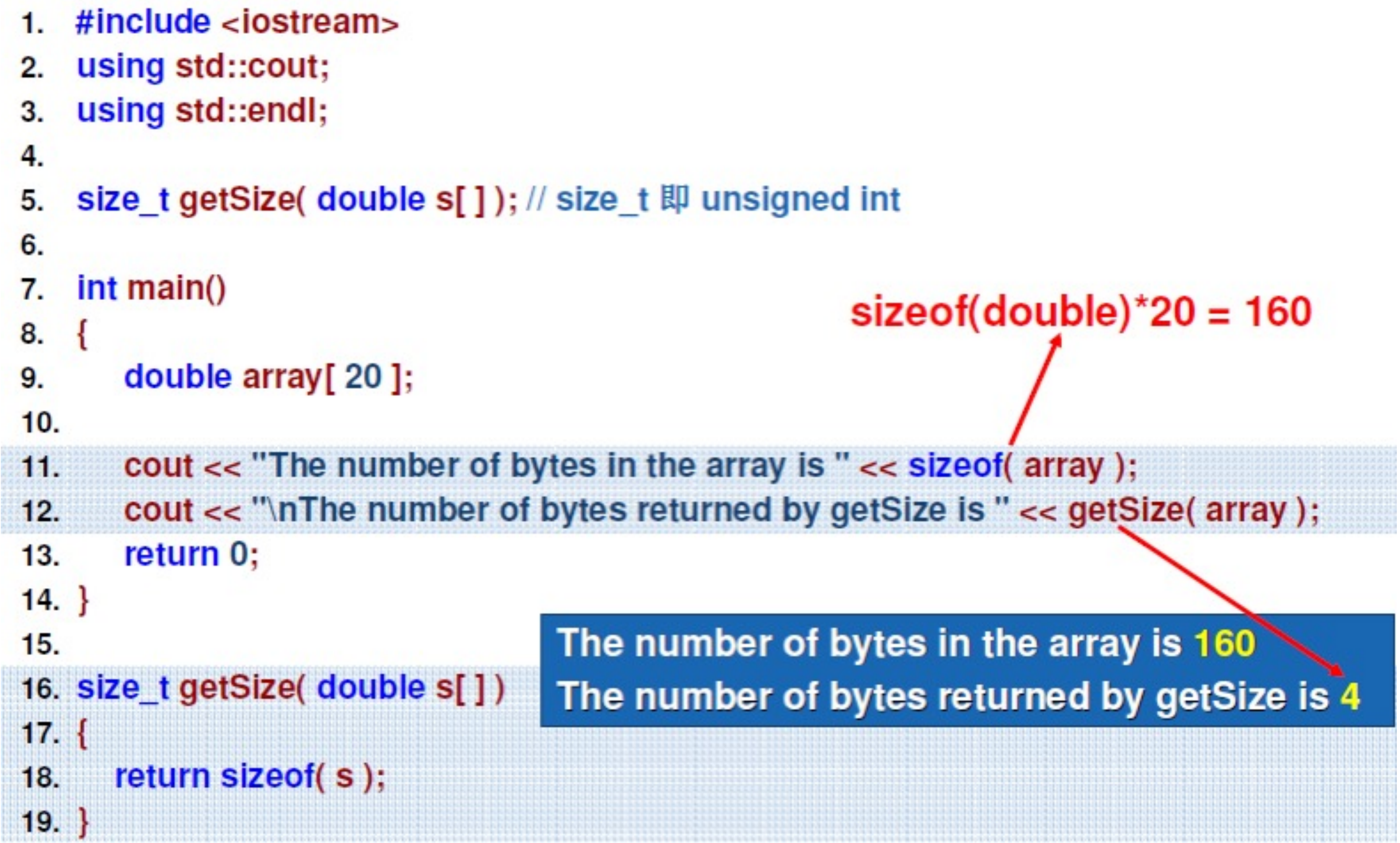
```
int array[20];
cout << sizeof(array); // 80 = 4 * 20
```

- Calculate number of elements

  sizeof(数组名)/sizeof(数组元素类型)

  sizeof(array) / sizeof(int) // 80 / 4 = 20

8.3 sizeof Operators



```
#include <iostream>
using std::cout;
using std::endl;

size_t getSize( double s[ ] ); // size_t 即 unsigned int

int main()
{
    double array[ 20 ];

    cout << "The number of bytes in the array is " << sizeof( array );
    cout << "\nThe number of bytes returned by getSize is " << getSize( array );
    return 0;
}

size_t getSize( double s[ ] )
{
   return sizeof( s );
}
```

sizeof(double)*20 = 160

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

8.3 sizeof Operators

- 数组名的值即数组首元素的地址
- 当数组名作为实参传递时, 本质上是传递数组地址

- 编译器不区分接受指针参数的函数和接受一维数组名参数的函数

```
size_t getSize( double s[ ] );
size_t getSize( double *s);
```

Topics

- 8.1 Pointer and Pointer Parameter
- 8.2 Using const with Pointers
- 8.3 sizeof Operators
- **8.4 Selection Sort Using Pass-by-Reference**
- 8.5 Pointer Expressions and Pointer Arithmetic
- 8.6 Relationship Between Pointers and Arrays
- 8.7 Introduction to Pointer-Based String Processing
- 8.8 Arrays of Pointers

8.4 Selection Sort Using Pass-by-Reference

- ❑ Selection Sort(选择排序): 首先找出最小元素, 将其交换至数组0号位; 其次, 在剩余元素中找出最小元素, 将其交换至1号位; 以此类推.

- ❑ void swap(int * const e1, int * const e2)
- ❑ 指针常量, 指向的数据可修改, 指向的内存地址不能改

```
void swap(int *const e1, int *const e2)
{
   int hold=*e1;
   *e1=*e2;
   *e2=hold;
}
void selectionSort(int * const array, int size)
{
   int smallest;
   for(int i=0;i<size;i++)
   {
      smallest=i;
      for(int j=i+1;j<size;j++)
         if(array[j]<array[smallest])
            smallest=j;
      swap(&array[i],&array[smallest]);
   }
}
```

Topics

- ❑ 8.1 Pointer and Pointer Parameter
- ❑ 8.2 Using const with Pointers
- ❑ 8.3 sizeof Operators
- ❑ 8.4 Selection Sort Using Pass-by-Reference
- ❑ 8.5 Pointer Expressions and Pointer Arithmetic
- ❑ 8.6 Relationship Between Pointers and Arrays
- ❑ 8.7 Introduction to Pointer-Based String Processing
- ❑ 8.8 Arrays of Pointers

8.5 Pointer Expressions and Pointer Arithmetic

- Pointers are valid operands in arithmetic expressions(算术), assignment expressions(赋值) and comparison expressions(比较).

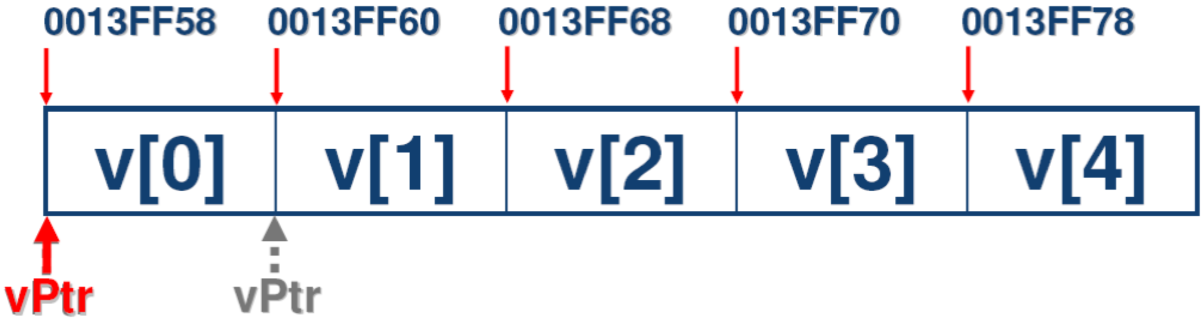
- However, not all the operators normally used in these expressions are valid with pointer variables.

- 指针运算一般与数组结合应用!

8.5 Pointer Expressions and Pointer Arithmetic

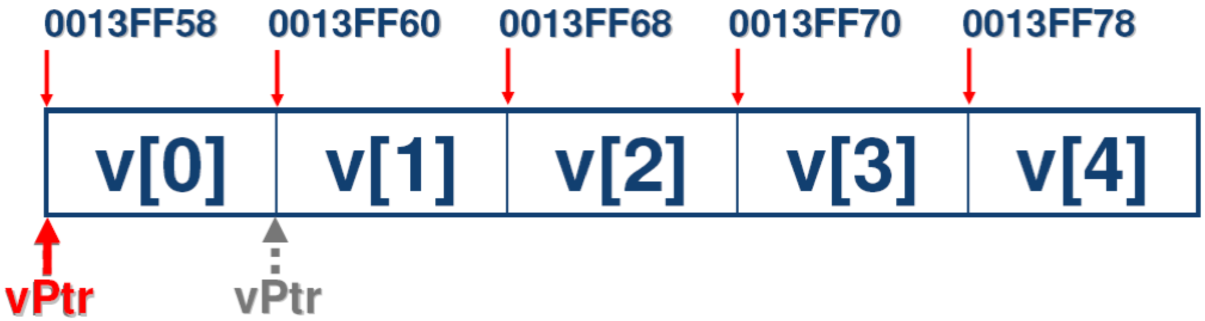
(1)自增、加法赋值运算　　// v[0]

```
double v[5] = {0};
double *vPtr = &v[0]; // 0013FF58
```



- ❑ vPtr++;　　// v[1]
- ❑ vPtr += 3;　// v[4]

8.5 Pointer Expressions and Pointer Arithmetic

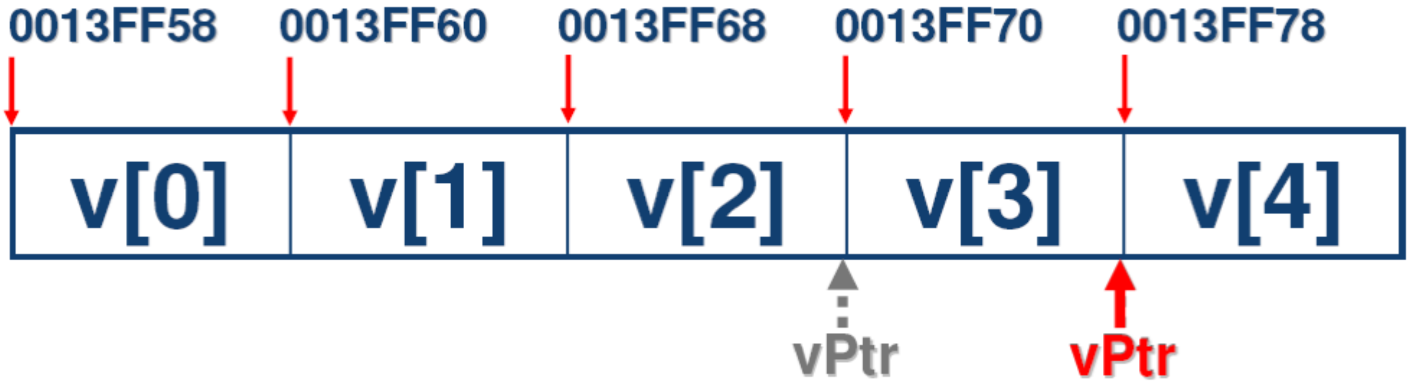


- ❑ 指针运算结论:
- ❑ 指针运算(假设指向类型type的指针)时, +/-n表示前移/后移n个元素, 其中n称为offset(偏移值)
- ❑ 从数值上看, 指针的值是加/减了n * sizeof(type)

8.5 Pointer Expressions and Pointer Arithmetic

(2)自减、减法赋值运算// v[4]

```
double v[5] = {0};
double *vPtr = &v[4]; // 0013FF78
```

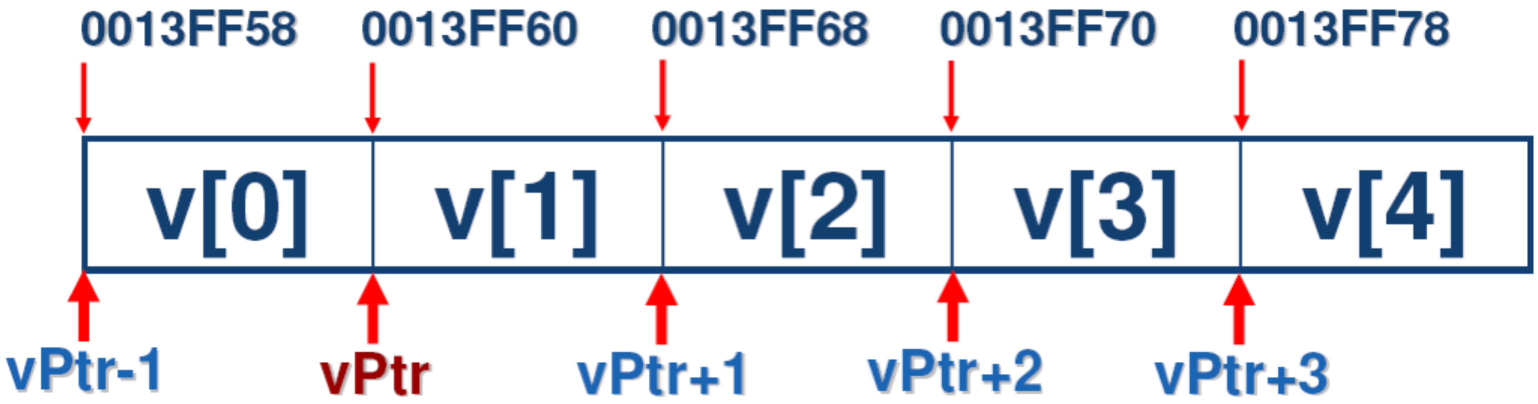


- ❑vPtr--;   // v[3]
- ❑vPtr -= 3; // v[0]

8.5 Pointer Expressions and Pointer Arithmetic

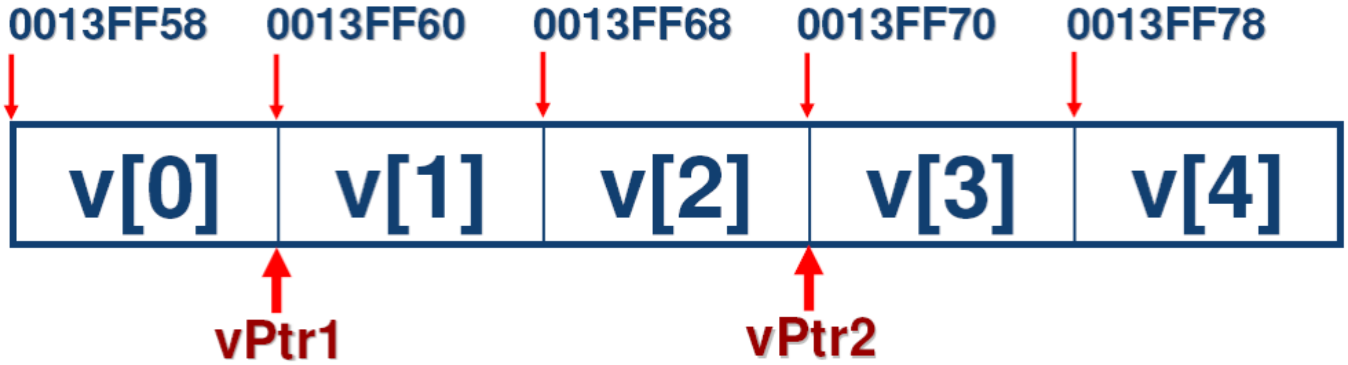
(3)加法、减法表达式// v[1]

```
double v[5] = {0};
double *vPtr = &v[1]; // 0013FF60
```



8.5 Pointer Expressions and Pointer Arithmetic

- ❑ double v[5] = {0};
- ❑ double *vPtr1 = &v[1]; // 0013FF60
- ❑ double *vPtr2 = &v[3]; // 0013FF70



- ❑ int x = vPtr2 – vPtr1; cout << x;
- ❑ 输出2, 表示两个指针间相差几个元素

8.5 Pointer Expressions and Pointer Arithmetic

(4)赋值运算和通用指针(Generic pointer)

- 仅有相同类型的指针之间可以进行赋值操作(否则必须进行类型转换), 特例: 通用指针void *

```
int num = 0;
int *ptrNum = &num;
void *p = ptrNum;
```

- 任意类型指针均可以赋值给通用指针, 反之不成立!
- 通用指针仅用于保存地址值, 不能进行解引用和算术运算

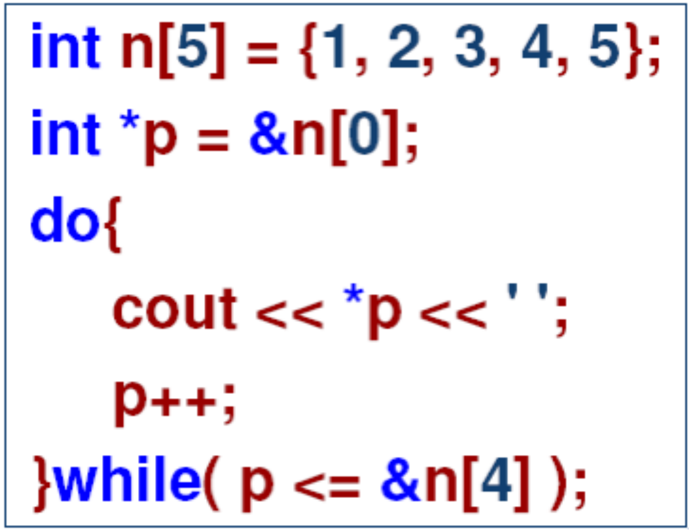
8.5 Pointer Expressions and Pointer Arithmetic

(5)等价与关系运算

❑ 等价运算符, 判断某指针是否为空指针

```
if (pGradeBook == 0) // NULL
        cout << “error” << endl;
else
        (*pGradeBook).displayMessage();
```

❑ 关系运算符, 一般用于数组

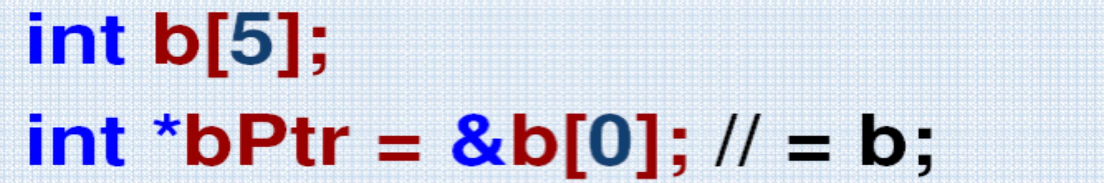


```
int n[5] = {1, 2, 3, 4, 5};
int *p = &n[0];
do{
    cout << *p << ' ';
    p++;
}while( p <= &n[4] );
```

Topics

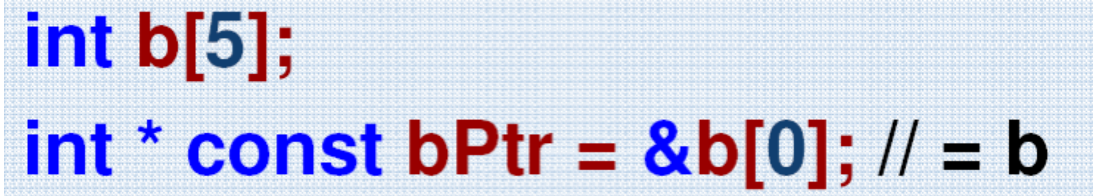
- 8.1 Pointer and Pointer Parameter
- 8.2 Using const with Pointers
- 8.3 sizeof Operators
- 8.4 Selection Sort Using Pass-by-Reference
- 8.5 Pointer Expressions and Pointer Arithmetic
- 8.6 Relationship Between Pointers and Arrays
- 8.7 Introduction to Pointer-Based String Processing
- 8.8 Arrays of Pointers

8.6 Relationship Between Pointers and Arrays



```
int b[5];
int *bPtr = &b[0]; // = b;
```

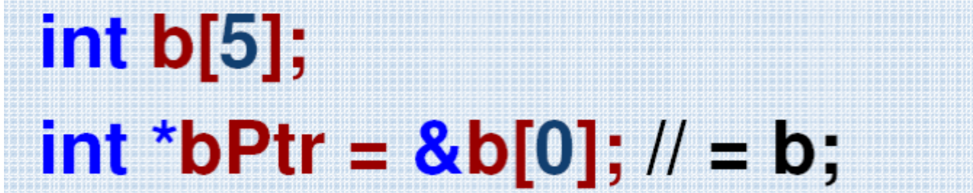
- ❑数组名b是指向数组首元素的指针常量
- ❑bPtr是指向数组首元素的指针



```
int b[5];
int * const bPtr = &b[0]; // = b
```

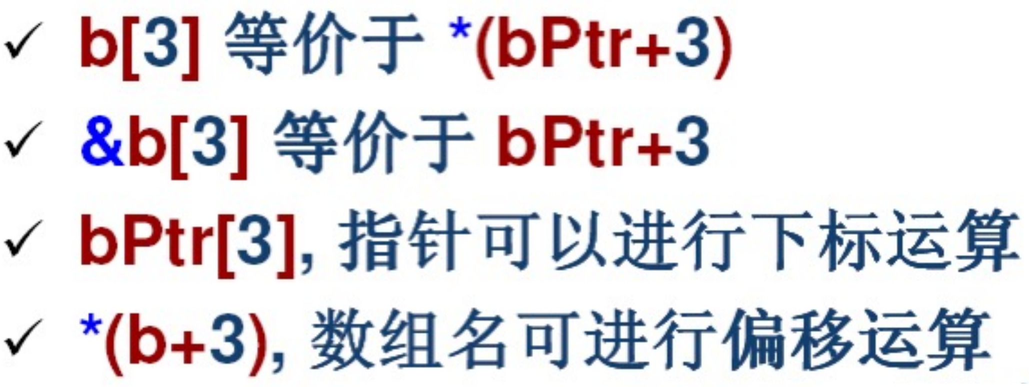
- ❑b实际上等价于此处的指针常量bPtr

8.6 Relationship Between Pointers and Arrays



```
int b[5];
int *bPtr = &b[0]; // = b;
```

- ❑ 除了b的const限定外, b和bPtr可互换使用:
- ❑ 数组subscript下标运算(方括号[ ]运算符)
- ❑ 指针offset偏移运算(指针算术运算)



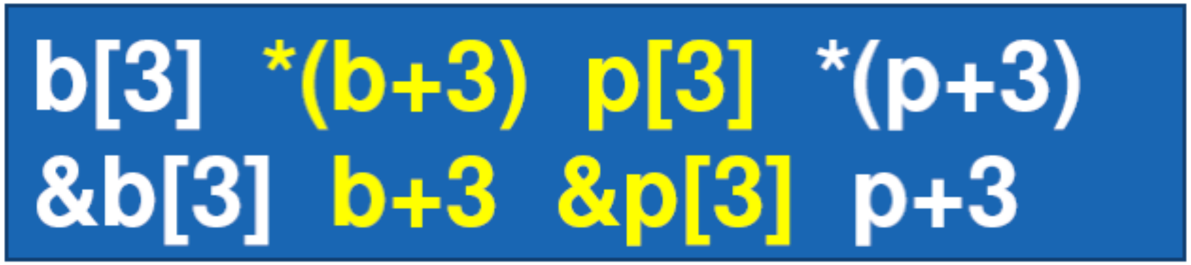
- ✓ b[3] 等价于 *(bPtr+3)
- ✓ &b[3] 等价于 bPtr+3
- ✓ bPtr[3], 指针可以进行下标运算
- ✓ *(b+3), 数组名可进行偏移运算

Q & A

```
     int nums[ ] = {1, 2, 3, 4, 5}; // 0013FF24
void * p0 = nums;
     cout << p0 << endl;
     const int *p2 = &nums[1];
     cout << ++p2 << endl;
     cout << *p2 << endl;
     *p2 = 9;      ✗
     cout << *p0 << endl;  ✗
     nums++;   ✗
     cout << *nums << endl;
```

Q & A

- ❑ 2. 简述数组名b和指针p的关系, 并多种方式表示数组元素b[3]及其地址.
- ❑ int b[5];
- ❑ int *p = &b[0];



```
b[3]  *(b+3)  p[3]  *(p+3)
&b[3]  b+3  &p[3]  p+3
```

Topics

- □ **8.1 Pointer and Pointer Parameter**
- □ **8.2 Using const with Pointers**
- □ **8.3 sizeof Operators**
- □ **8.4 Selection Sort Using Pass-by-Reference**
- □ 8.5 Pointer Expressions and Pointer Arithmetic
- □ 8.6 Relationship Between Pointers and Arrays
- □ 8.7 Introduction to Pointer-Based String Processing
- □ 8.8 Arrays of Pointers

8.7 Introduction to Pointer-Based String Processing

- ❑ Part I: Fundamentals of Characters and Pointer-Based Strings

- ❑ Part II: String Manipulation Functions of the String-Handling Library

8.7 Introduction to Pointer-Based String Processing

- **Characters**字符
  - ‘A’, ‘+’, ‘?’, ‘\0’, ‘\n’ , ‘\\’ 等
  - 字符常量, 值为其ASCII 码值

- **char**类型变量可以直接进行关系运算

```
char a = ‘a’, b;
cin >> b;
if ( a > b )
    cout << a << endl;
else
    cout << b << endl;
```

8.7 Introduction to Pointer-Based String Processing

- ❑ String字符串
  - ❖include letters(字母A~Z, a-z), digits(数字0~9) and various special characters
  - ❖such as ‘+’, ‘-’, ‘*’, ‘/’ and ‘$’.
- ❑ “Welcome to C++!”, “Hello, World!” 等
  - ❖String literals字符串文本, or string constants字符串常量

8.7 Introduction to Pointer-Based String Processing

(1)字符数组与字符串的关系

- 数组中的元素可以是引用之外的任何类型,其中一种特殊类型即字符数组, 可用于表示字符串!
- C/C++语言中的字符串(*C-Style String*)

  连续内存区域+ 字符+ ‘\0’结尾

  其中‘\0’称为**NULL Char**, 即空字符, 对应整数值为 0

‘f’	‘i’	‘r’	‘s’	‘t’	‘\0’

8.7 Introduction to Pointer-Based String Processing

(2)字符数组的初始化

- 初始化列表(与普通数组相同)

```
char string1[ ] = { ‘f’, ‘i’, ‘r’, ‘s’, ‘t’, ‘\0’ };
```

- 通过字符串常量进行初始化

```
char string2[ ] = “first”;
```

8.7 Introduction to Pointer-Based String Processing

(3)字符数组的赋值

- 循环语句逐个元素赋值(与普通数组相同)
- 流操作运算

```
char string3[ 7 ];
cin >> string3; // Hello
```

- 将用户输入的字符串从string3对应的内存起始地址开始写入, 并在结尾处加空字符!

‘H’	‘e’	‘l’	‘l’	‘o’	‘\0’	3	‘A’

8.7 Introduction to Pointer-Based String Processing

-
  ```
  int n[20];
  for( int i = 0; i < 20; i++)
     cin >> n[ i ];
  ```

-
  ```
  char s[20];
  cin >> s;
  ```
- 若输入多于19个字符, 则溢出
- 遇空白字符认为输入结束

8.7 Introduction to Pointer-Based String Processing

- char s[20];
- cin.getline(s, n);   // n = 20

  cin.getline(s, n, '\n'); // 缺省实参
- The function stops reading characters
  - 当读到delimiter character(分隔符, 缺省为'\n')
  - 或已读到n-1 个字符(防止溢出)
- cin >> setw(n) >> s;   // n = 20
- 指定最多读入n-1个字符, 并在尾部自动添加null character

7.4 String and Char Array

(4)字符数组的输出

- 逐个元素读取并输出

- 流操作运算

  cout << string3; // char string3[8]

  从string3对应的内存起始地址开始读数据,直至遇到空字符

‘H’	‘e’	‘l’	‘l’	‘o’	‘!’	‘!’	‘\0’

8.7 Introduction to Pointer-Based String Processing

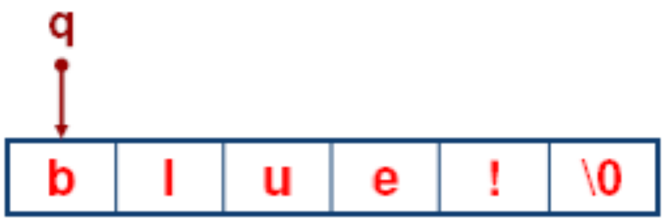
```
1. char q[ ] = "blue!";
2. q[2] = 'A';
3. cout << q << endl;
```

q[0]	q[1]	q[2]	q[3]	q[4]	q[5]
b	l	A	e	!	\0

- ❑ 用字符串常量对数组初始化

const

```
1.  char *q = "blue!";
2.  *(q+2) = 'A';
3.  cout << q << endl;
```



- ❑ 将字符串常量的地址赋值给字符指针
- ❑ 赋值‘A’时报内存访问错误

Q & A

	非char数组 int num[6]	char数组 char s[6]
1. 初始化	初始化列表 int num[6] = {1, 2, 3};	初始化列表 char s[6] = {‘1’, ‘2’, ‘3’};
作为 String处理		字符串常量 char s[6] = “Hello”;
2. 赋值	逐个元素 Repetition Structure	逐个元素 Repetition Structure

作为
String处理
		cin >> s;
		cin.getline(s, 6);

3. 输出	逐个元素 Repetition Structure	逐个元素 Repetition Structure
作为 String处理		cout << s;

Q & A

- 1. 给定字符串，将其中的大写字母改为对应的小写字母，小写字母改为对应的大写字母。
- 例：“Hello!”　　改写后为 “hELLO!”
- 函数原型　　void change(char * s);

- 2. 给定字符串，返回该字符串第一个只出现一次的字符。
- 例：“AbcAcb!ello! ”　返回 ‘e’
- 函数原型　　char firstC(const char * s);

8.7 Introduction to Pointer-Based String Processing

- ❑ **Part I: Fundamentals of Characters and Pointer-Based Strings**

- ❑ **Part II: String Manipulation Functions of the String-Handling Library 22.8**
- ❑ **C语言继承而来的String处理函数库头文件 <cstring>**

8.7 Introduction to Pointer-Based String Processing

- char *strcpy(char *s1, const char *s2);
- char *strncpy(char *s1, const char *s2, size_t n);
- char *strcat(char *s1, const char *s2);
- char *strncat(char *s1, const char *s2, size_t n);
- int strcmp(const char *s1, const char *s2);
- int strncmp(const char *s1, const char *s2, size_t n
- char *strtok(char *s1, const char *s2);
- size_t strlen(const char *s);

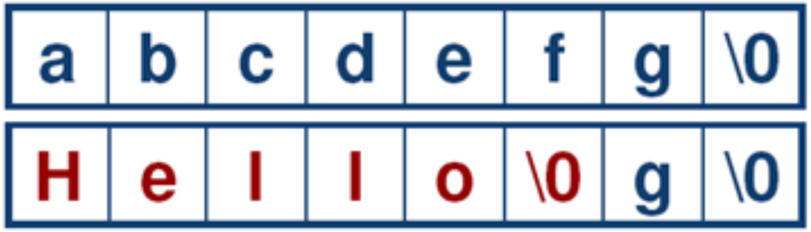
8.7 Introduction to Pointer-Based String Processing

(1) char *strcpy(char *s1, const char *s2);

- Copies the string s2 into the character array s1. The value of s1 is returned.

- char s1[8]:

  “Hello”:



a	b	c	d	e	f	g	\0
H	e	l	l	o	\0	g	\0

- 调用：cout<<strcpy(s1, “Hello”);

8.7 Introduction to Pointer-Based String Processing

(2)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.
- Note: strncpy并不保证拷贝null character, 仅当n的值大于s2的长度时null character才会拷贝.

```
#include <cstring> // Fig_8.31
using std::strcpy;
using std::strncpy;
int main()
{
    char x[ ] = "Happy Birthday to You"; // string length 21
    char y[ 25 ], 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';

    // 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;
}
```

```
x is: Happy Birthday to You
y is: Happy Birthday to You
z is: Happy Birthday
```

8.7 Introduction to Pointer-Based String Processing

(3)char *strcat(char *s1, const char *s2);

- ❑ Appends s2 to s1. The first character of s2 overwrites s1’ the terminating null character.

  s1 + s2 + null character

(4)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.

  s1 + n char of s2 + null character

```
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 (lengtl
```

```
s1 = Happy New Year
s2 = New Year
```

```
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 chai
```

```
s1 = Happy New Year
s3 = Happy
```

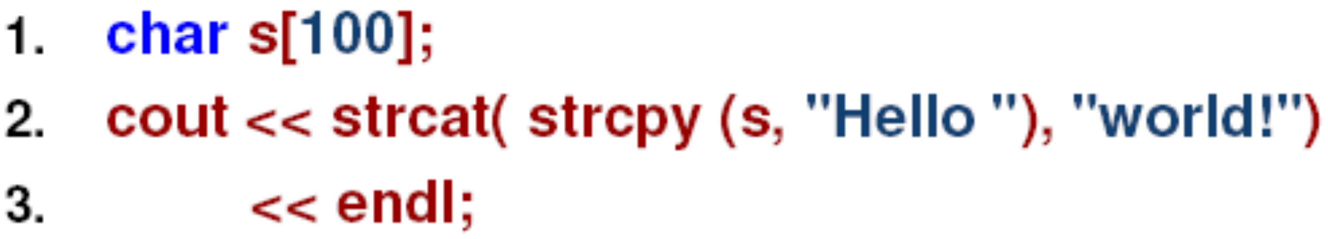
```
cout << "\n\nAfter strncat(s3, s1, 6):\ns1 = " << s1
     << "\ns3 = " << s3;

strcat( s3, s1 ); // concatenate s1 to s3
```

```
s1 = Happy New Year
s3 = Happy Happy New Year
```

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

8.7 Introduction to Pointer-Based String Processing



```
char s[100];
cout << strcat( strcpy (s, "Hello "), "world!")
          << endl;
```

- ❑ **Hello world!**

8.7 Introduction to Pointer-Based String Processing

(5)int strcmp(const char *s1, const char *s2);

❑ Compares the string s1 with the string s2. The function returns a value of:

① =0: s1 is equal to s2

② <0 (usually -1): s1 is less than s2

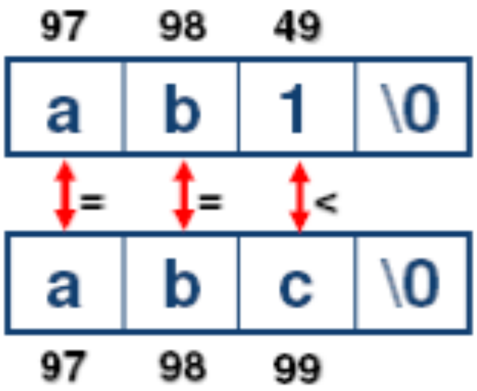
③ >0 (usually 1): s1 is greater than s2.

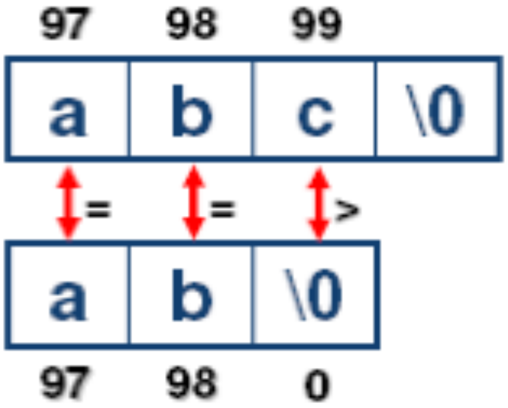
(6)int strncmp(const char *s1, const char *s2, size_t n);

❑ Compares up to n characters of the string s1 with the string s2.

8.7 Introduction to Pointer-Based String Processing

- □ strcmp(“ab1”, “abc”) 结果为-1
- □ strncmp(“ab1”, “abc”, 2) 结果为0
- □ strcmp(“abc”, “ab”) 结果为1





8.7 Introduction to Pointer-Based String Processing

(7)size_t strlen(const char *s);

- Determines the length of string s. The number of characters preceding the terminating null character is returned.

8.7 Introduction to Pointer-Based String Processing

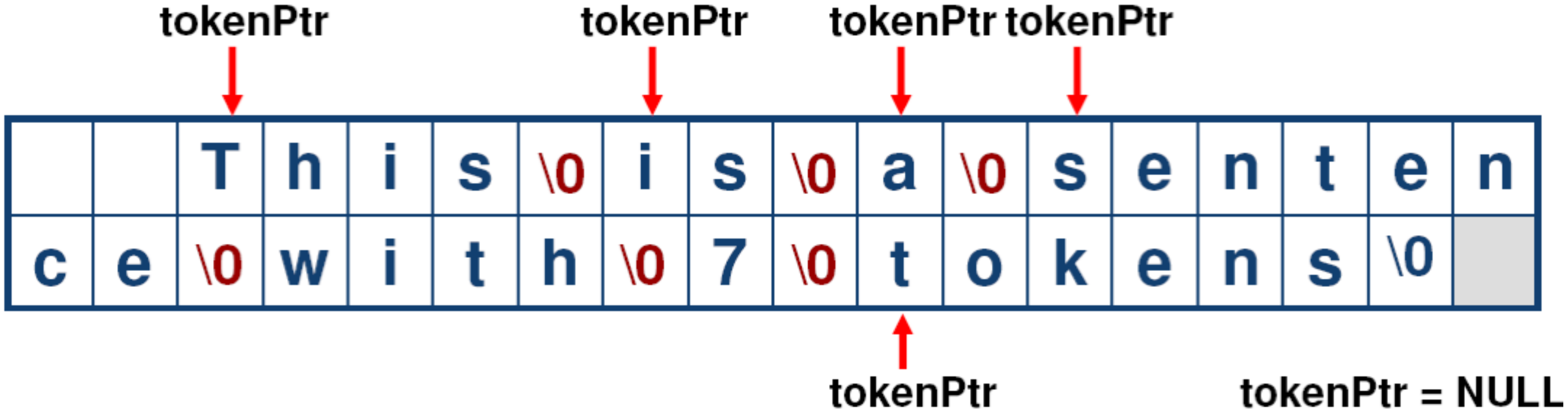
(8)char *strtok(char *s1, const char *s2);

- 将字符串s2中的字母作为delimiter分隔符, 将字符串s1分解为若干个token.
- char sentence[ ]= “ This is a sentence with 7 tokens”;
- 1. tokenPtr = strtok(sentence, “ ”);
- 2. tokenPtr = strtok(NULL, “ ”);

8.7 Introduction to Pointer-Based String Processing

(8)char *strtok(char *s1, const char *s2);

- ❑ tokenPtr = strtok(sentence, “ ”);
- ❑ tokenPtr = strtok(NULL, “ ”);



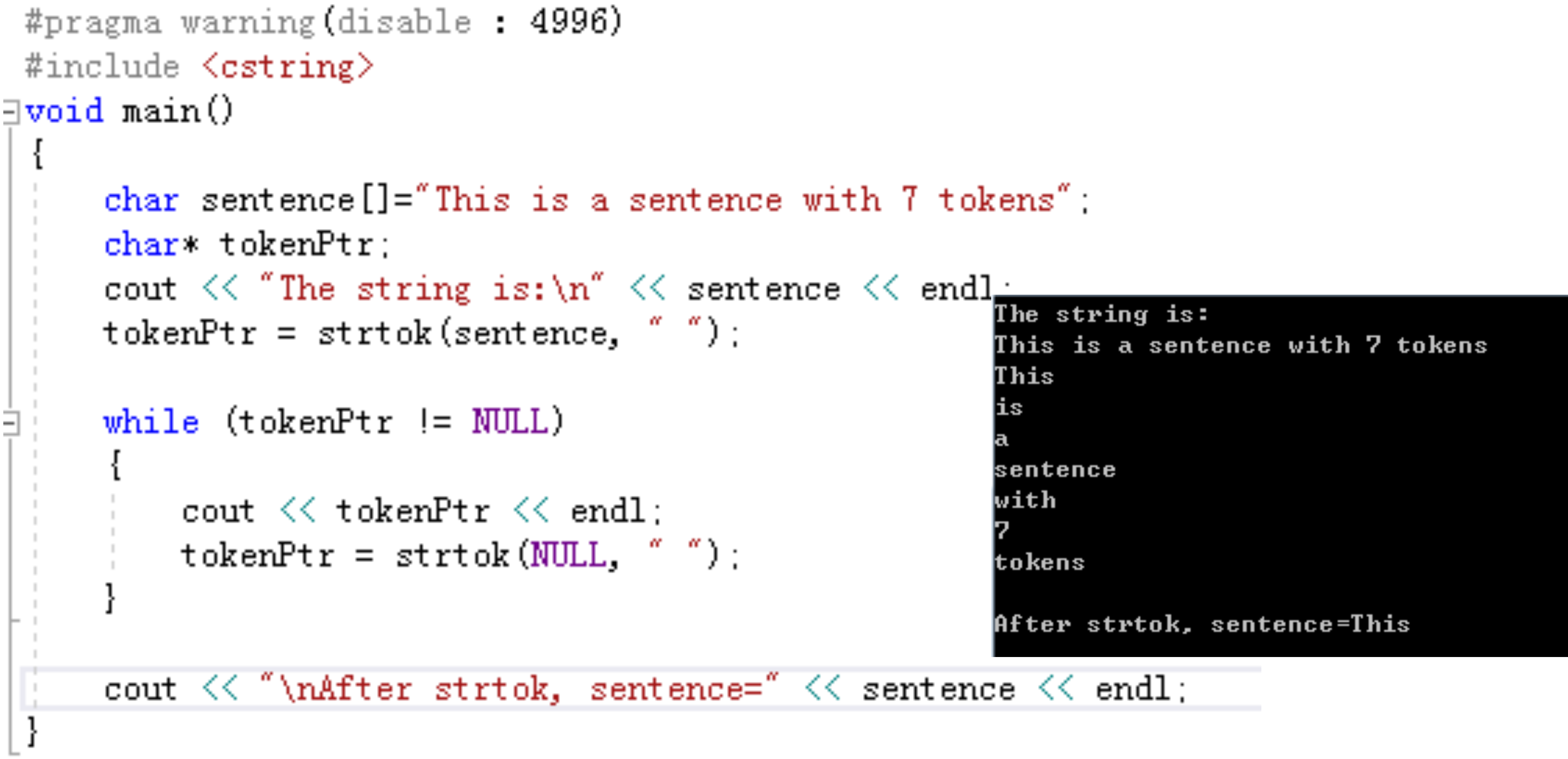
tokenPtr tokenPtr tokenPtr tokenPtr

		T	h	i	s	\0	i	s	\0	a	\0	s	e	n	t	e	n
c	e	\0	w	i	t	h	\0	7	\0	t	o	k	e	n	s	\0	

tokenPtr

tokenPtr = NULL

8.7 Introduction to Pointer-Based String Processing



```
#pragma warning(disable : 4996)
#include <cstring>
void main()
{
    char sentence[]="This is a sentence with 7 tokens";
    char* tokenPtr;
    cout << "The string is:\n" << sentence << endl;
    tokenPtr = strtok(sentence, " ");

    while (tokenPtr != NULL)
    {
        cout << tokenPtr << endl;
        tokenPtr = strtok(NULL, " ");
    }

    cout << "\nAfter strtok, sentence=" << sentence << endl;
}
```

```
The string is:
This is a sentence with 7 tokens
This
is
a
sentence
with
7
tokens

After strtok, sentence=This
```

8.7 Introduction to Pointer-Based String Processing

- □+国际区号-(区号) 本地号码
- □ char s[ ] = "+86-(025) 52091012";

- □1. strtok(s, "+-() ");
- □2. strtok(NULL, "+-() ");

Topics

- **8.1 Pointer and Pointer Parameter**
- **8.2 Using const with Pointers**
- **8.3 sizeof Operators**
- **8.4 Selection Sort Using Pass-by-Reference**
- 8.5 Pointer Expressions and Pointer Arithmetic
- 8.6 Relationship Between Pointers and Arrays
- 8.7 Introduction to Pointer-Based String Processing
- 8.8 Arrays of Pointers

8.8 Arrays of Pointers

❑数组元素可以是除引用外的任意类型:

```
int m1 = 11, m2 = 22, m3 = 33;
int *ms[ ] = { &m1, &m2, &m3 };
cout << *ms[0] << ' '
        << *ms[1] << ' '
        << *ms[2]
        << endl;
```

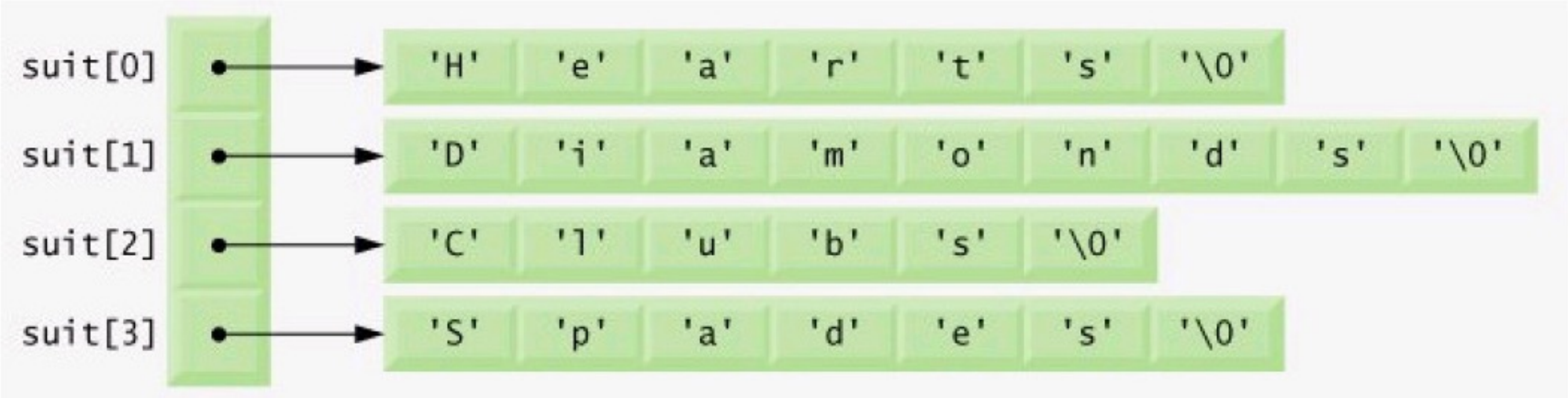
8.8 Arrays of Pointers

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

可以省略，但一般保留！

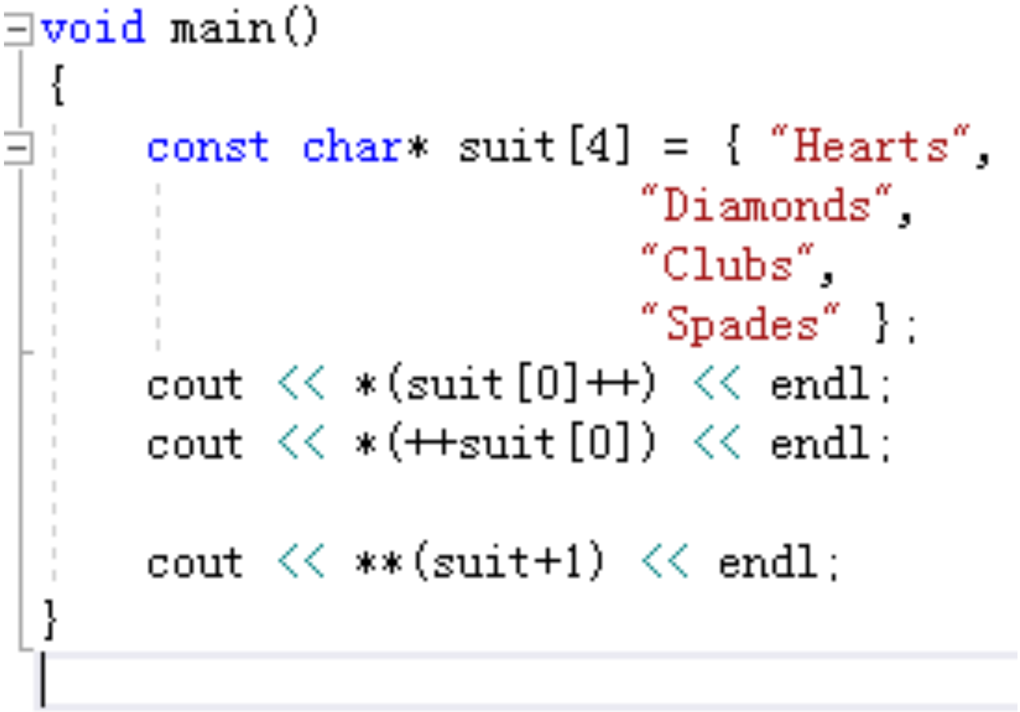
❑ String array,字符串数组

值为首字符的地址



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'

8.8 Arrays of Pointers



```
void main()
{
    const char* suit[4] = { “Hearts”,
                            “Diamonds”,
                            “Clubs”,
                            “Spades” };
    cout << *(suit[0]++) << endl;
    cout << *(++suit[0]) << endl;

    cout << **(suit+1) << endl;
}
```

- H
- a
- D

Summary

- 指针Pointer
- sizeof运算符
- 指针表达式和指针运算
- const pointer指针常量和pointer to const
- 指针和数组的关系
- 指针数组
- 常用的基于指针字符串的处理函数

Homework

- □实验必做题目：
- □实验手册Ex2，Ex3，Ex4。