第四讲 数组与vector

学习目标：

- 声明数组、初始化数组
- 引用数组中的元素
- 将数组传递给函数
- 使用C++标准库类模板array
- 使用C++标准库类模板 vector

4-1 数组的创建与使用

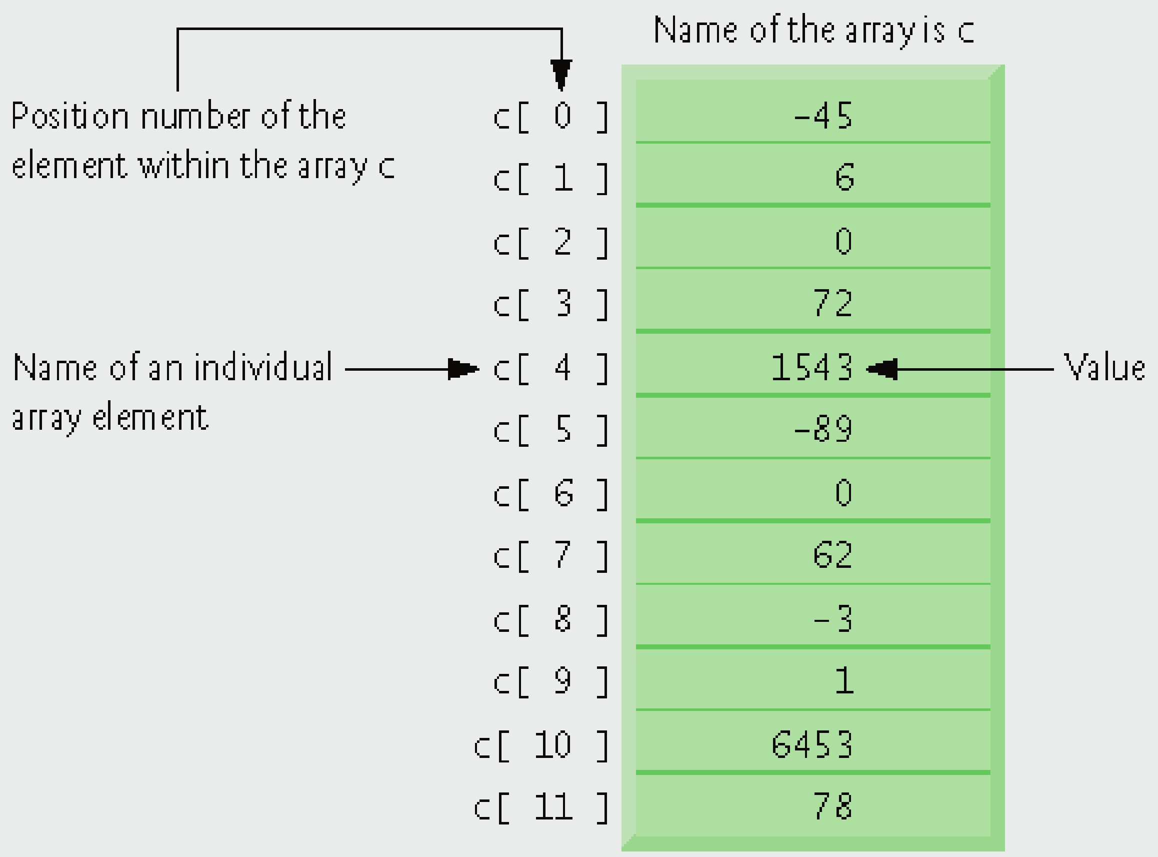
- 数组的创建
- 数组的初始化
- 数组的遍历

1 Introduction

- Arrays(数组)
  - 包含同一数据类型的数据结构
  - 占用一段连续的内存空间
  - 创建后大小不能改变
  - 通过索引的方式访问数组中的元素

1 Introduction

```
int  c[12];
```



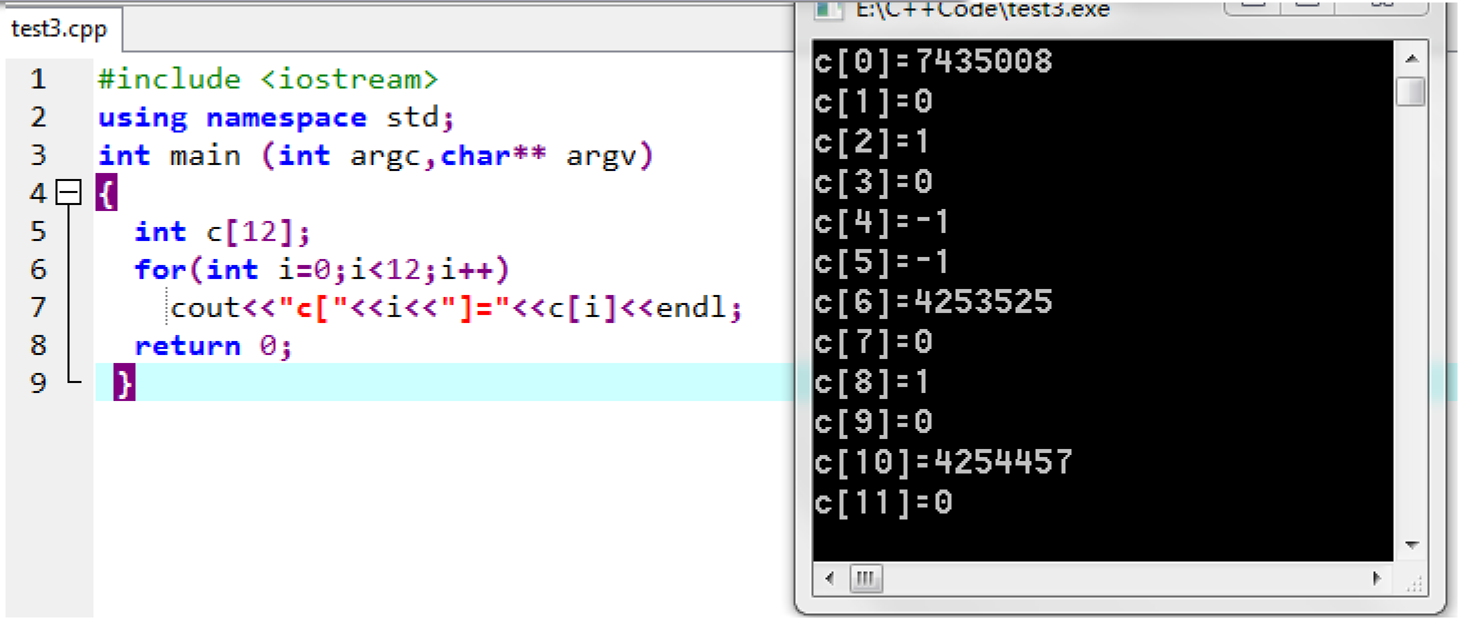
2 Declaring and Initializing Arrays

- Declaring an array(数组声明)
  - 类型、数组名、数组大小

    如：int c[12];
  - 数组大小为大于 0 的常整数

2 Declaring and Initializing Arrays

数组为什么要初始化？



test3.cpp

```cpp
#include <iostream>
using namespace std;
int main (int argc,char** argv)
{
   int c[12];
   for(int i=0;i<12;i++)
     cout<<"c["<<i<<"]="<<c[i]<<endl;
   return 0;
 }
```

E:\C++Code\test3.exe

```
c[0]=7435008
c[1]=0
c[2]=1
c[3]=0
c[4]=-1
c[5]=-1
c[6]=4253525
c[7]=0
c[8]=1
c[9]=0
c[10]=4254457
c[11]=0
```

可以看出，数组没有初始化之前，各元素取一些随机值，如果直接使用，可能会产生错误的结果。因此，使用之前应该初始化。

数组初始化方法

（1）循环法初始化数组成员

```
int n[ 10 ];

for ( int i = 0; i < 10; i++ )
    n[ i ] = 0;
```

数组初始化方法

（2）用初始化列表来初始化数组成员

例： int n[] = { 10, 20, 30, 40, 50 };

- 如果初始化列表的数据量小于数组长度，其余数组元素将被初始化为 0

例： int n[10] = { 8 };

- 如果初始化列表的数据量大于数组长度，产生编译错误

C++ How to Program

```
// Fig. 7.3: fig07_03.cpp
// Initializing an array.
#include <iostream>
using std::cout;
using std::endl;

#include <iomanip>
using std::setw;

int main()
{
   int n[ 10 ]; // n is an array of 10 integers

   // initialize elements of array n to 0
   for ( int i = 0; i < 10; i++ )
      n[ i ] = 0; // set element at location i to 0

   cout << "Element" << setw( 13 ) << "Value" << endl;
```

Declare **n** as an array of **int**s with 10 elements

Each **int** initialized is to **0**

```
   // output each array element's value
   for ( int j = 0; j < 10; j++ )
      cout << setw( 7 ) << j << setw( 13 ) << n[ j ] << endl;

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

n[j] returns **int** associated with index **j** in array **n**

```
Element        Value
      0            0
      1            0
      2            0
      3            0
      4            0
      5            0
      6            0
      7            0
      8            0
      9            0
```

Each **int** has been initialized to **0**

setw(int n), setfill(char c)

- setw(n)：预设宽度。规定其后内容所占宽度。

  如：cout << setw(5) << 255 << endl;

  结果是： (空格)(空格)255

- setfill(char c): 在预设宽度中如果存在没用完的宽度，则用设置的字符 c 填充

  如 cout << setfill('@') << setw(5) << 255 << endl;

  结果是： @@255

只对其后的内容有效！

setbase(int n), setprecision(int n)

- setbase(int n) ： 将数字转换为 n 进制

如
```
cout << setbase(8)  << setw(5) << 255 << endl;
cout << setbase(10) << setw(5) << 255 << endl;
cout << setbase(16) << 255 <<endl;
```
结果是:
```
(空格)(空格)377
(空格)(空格)255
(空格)(空格)ff
```
最后一行没有新的setw()

- setprecision(int n)：控制输出流显示浮点数的数字个数。C++默认的流输出数值有效位是6。
- 如果setprecision(n)与setiosflags(ios::fixed)合用，可以控制小数点右边的数字个数。setiosflags(ios::fixed)是用定点方式表示实数。
- 如果与setiosnags(ios::scientific)合用， 可以控制指数表示法的小数位数。setiosflags(ios::scientific)是用指数方式表示实数。

3 Examples Using Arrays

- Initializing an array in a declaration with an initializer list (Cont.)
  - 如果初始化列表中的数据个数比数组元素少
    - ✓ Remaining elements are initialized to zero
    - ✓ Example

      ```
      int n[ 10 ] = { 8 };
      ```

      - ◇ Explicitly(显式) initializes first element to eight
      - ◇ Implicitly(隐式) initializes remaining nine elements to zero
  - 如果初始化列表中的数据个数比数组元素多
    - ✓ Compilation error

```
// Fig. 7.4: fig07_04.cpp
// Initializing an array in a declaration.
#include <iostream>
using std::cout;
using std::endl;

#include <iomanip>
using std::setw;

int main()
{
   // use initializer list to initialize array n
   int n[ 10 ] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };

   cout << "Element" << setw( 13 ) << "Value" << endl;
```

Declare **n** as an array of **int**s

Compiler uses initializer list to initialize array

```

   // output each array element's value
   for ( int i = 0; i < 10; i++ )
      cout << setw( 7 ) << i << setw( 13 ) << n[ i ] << endl;

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

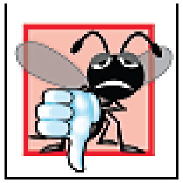
```
Element        Value
      0           32
      1           27
      2           64
      3           18
      4           95
      5           14
      6           90
      7           70
      8           60
      9           37
```

3 Examples Using Arrays

- Constant variables(常变量)
  - const 修饰符，又称为常变量或只读变量
  - 声明时必须进行初始化，且以后不能修改
  - 使用常变量来声明数组长度使程序更加灵活，避免了 “magic numbers”

3 Examples Using Arrays

性能提示：假如不是在执行时用赋值语句来初始化数组，而是在编译时用一个数组初始化列表来初始化数组，程序执行速度会更快。



常见编程错误：只有常变量才可用于声明自动和静态数组的长度。不用常变量会造成语法错误。

C++ How to Program

```
1  // Fig. 7.5: fig07_05.cpp
2  // Set array s to the even integers from 2 to 20.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include <iomanip>
8  using std::setw;
9
10 int main()
11 {
12    // constant variable can be used to specify array size
13    const int arraySize = 10;
14
15    int s[ arraySize ]; // array s has 10
16
17    for ( int i = 0; i < arraySize; i++ ) // set the values
18       s[ i ] = 2 + 2 * i;
```

Declare constant variable **`arraySize`** using the **`const`** keyword

Declare array that contains 10 **`int`**s

第13行可改为：**int arraySize = 10** ？

Use array index to assign element's value

```cpp

   cout << "Element" << setw( 13 ) << "Value" << endl;

   // output contents of array s in tabular format
   for ( int j = 0; j < arraySize; j++ )
      cout << setw( 7 ) << j << setw( 13 ) << s[ j ] << endl;

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

```
Element        Value
      0            2
      1            4
      2            6
      3            8
      4           10
      5           12
      6           14
      7           16
      8           18
      9           20
```

```
// Fig. 7.7: fig07_07.cpp
// A const variable must be initialized.

int main()
{
   const int x; // Error: x must be initialized

   x = 7; // Error: cannot modify a const variable

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

Must initialize a constant at the time of declaration

Cannot modify a constant

Borland C++ command-line compiler error message:

```
Error E2304 fig07_07.cpp 6: Constant variable 'x' must be initialized
   in function main()
Error E2024 fig07_07.cpp 8: Cannot modify a const object in function main()
```

Microsoft Visual C++.NET compiler error message:

```
C:\cpphtp5_examples\ch07\fig07_07.cpp(6) : error C2734: 'x' : const object
   must be initialized if not extern
C:\cpphtp5_examples\ch07\fig07_07.cpp(8) : error C2166: l-value specifies
   const object
```

GNU C++ compiler error message:

```
fig07_07.cpp:6: error: uninitialized const `x'
fig07_07.cpp:8: error: assignment of read-only variable `x'
```

Error messages differ based on the compiler

```
// Fig. 7.9: fig07_09.cpp
// Bar chart printing program.
#include <iostream>
using std::cout;
using std::endl;

#include <iomanip>
using std::setw;

int main()
{
   const int arraySize = 11;
   int n[ arraySize ] = { 0, 0, 0, 0, 0, 0, 1, 2, 4, 2, 1 };

   cout << "Grade distribution:" << endl;

   // for each element of array n, output a bar of the chart
   for ( int i = 0; i < arraySize; i++ )
   {
      // output bar labels ("0-9:", ..., "90-99:", "100:" )
      if ( i == 0 )
         cout << "  0-9: ";
      else if ( i == 10 )
         cout << "  100: ";
      else
         cout << i * 10 << "-" << ( i * 10 ) + 9 << ": ";
```

每行中的 * 个数

Declare **array** with initializer list

```cpp

      // print bar of asterisks
      for ( int stars = 0; stars < n[ i ]; stars++ )
         cout << '*';

      cout << endl; // start a new line of output
   } // end outer for

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

For each array element, print the associated number of asterisks

```
Grade distribution:
  0-9:
10-19:
20-29:
30-39:
40-49:
50-59:
60-69: *
70-79: **
80-89: ****
90-99: **
  100: *
```

C++ has no array bounds checking. Does not prevent the computer from referring to an element that does not exist. Could lead to serious execution-time errors!

3 Examples Using Arrays

- 用字符数组来存储和处理字符串
  - char string1[] = { 'f', 'i', 'r', 's', 't', '\0' };
  - char string1[]="first"; -- 正确！
  - char string1[]='first'; -- 错误！

- Can also input a string directly into a character array from the keyboard using `cin >>`

```
cin >> string1;
```

  - ✓ `cin >>` may read more characters than the array can store, may cause other errors

- A character array representing a null-terminated string can be output with `cout <<`

（以’\0’结尾的字符数组可以通过 cout << 进行输出）

```
cout<< string1;
```

```
// Fig. 7.12: fig07_12.cpp
// Treating character arrays as strings.
#include <iostream>
using std::cout;
using std::cin;
using std::endl;

int main()
{
   char string1[ 20 ]; // reserves 20 characters
   char string2[] = "string literal"; // reserves 15 characters

   // read string from user into array string1
   cout << "Enter the string \"hello there\": ";
   cin >> string1; // reads "hello" [space terminates input]

   // output strings
   cout << "string1 is: " << string1 << "\nstring2 is: " << string2;

   cout << "\nstring1 with spaces between characters is:\n";

```

Store **"string literal"** as an array of characters

Initializing an array of characters using **cin**

Output array using **cin**

```
   // output characters until null character is reached
   for ( int i = 0; string1[ i ] != '\0'; i++ )
      cout << string1[ i ] << ' ';

   cin >> string1; // reads "there"
   cout << "\nstring1 is: " << string1 << endl;

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

Loop until the terminating null character is reached

Accessing specific characters in the array

```
Enter the string "hello there": hello there
string1 is: hello
string2 is: string literal
string1 with spaces between characters is:
h e l l o
string1 is: there
```

4 Passing Arrays to Functions

- 向被调用函数传递数组名作为实际参数
  - 定义数组：int hourlyTemperatures[24];
  - 函数调用：modifyArray(hourlyTemperatures, 24);
- 接收数组作为参数的函数原型
  - 函数原型：void modArray(int b[], int arraySize);

一般是两个参数，一个数组名，另一个数组长度

```
// Fig. 7.14: fig07_14.cpp
// Passing arrays and individual array elements to functions.
#include <iostream>
using std::cout;
using std::endl;

#include <iomanip>
using std::setw;

void modifyArray( int [], int ); // appears strange
void modifyElement( int );

int main()
{
   const int arraySize = 5; // size of array a
   int a[ arraySize ] = { 0, 1, 2, 3, 4 }; // initialize array a

   cout << "Effects of passing entire array by reference:"
      << "\n\nThe values of the original array are:\n";

   // output original array elements
   for ( int i = 0; i < arraySize; i++ )
      cout << setw( 3 ) << a[ i ];

   cout << endl;

   // pass array a to modifyArray by reference
   modifyArray( a, arraySize );
   cout << "The values of the modified array are:\n";
```

Function takes an array as argument

函数原型只写参变量类型不写名

Declare **5-int** array **array** with initializer list

Pass entire array to function modifyArray

```

   // output modified array elements
   for ( int j = 0; j < arraySize; j++ )
      cout << setw( 3 ) << a[ j ];

   cout << "\n\n\nEffects of passing array element by value:"
      << "\n\na[3] before modifyElement: " << a[ 3 ] << endl;

   modifyElement( a[ 3 ] ); // pass array element a[ 3 ] by value
   cout << "a[3] after modifyElement: " << a[ 3 ] << endl;

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

// in function modifyArray, "b" points to the original array "a" in memory
void modifyArray( int b[], int sizeOfArray )
{
   // multiply each array element by 2
   for ( int k = 0; k < sizeOfArray; k++ )
      b[ k ] *= 2;
} // end function modifyArray
```

Pass array element **a[3]** to function **modifyElement**

Function **modifyArray** manipulates the array directly

```

// in function modifyElement, "e" is a local copy of
// array element a[ 3 ] passed from main
void modifyElement( int e )
{
   // multiply parameter by 2
   cout << "Value of element in modifyElement: " << ( e *= 2 ) << endl;
} // end function modifyElement
```

Function **modifyElement** manipulates array element's copy

```
Effects of passing entire array by reference:

The values of the original array are:
  0  1  2  3  4
The values of the modified array are:
  0  2  4  6  8


Effects of passing array element by value:

a[3] before modifyElement: 6
Value of element in modifyElement: 12
a[3] after modifyElement:
```

4 Passing Arrays to Functions

- const array parameters(函数参数定义为常变量)
  - const 修饰符
  - 阻止被调用的函数修改数组值
  - 在函数体内数组元素为常量
  - 防止程序员意外修改数组元素

```cpp
// Fig. 7.15: fig07_15.cpp
// Demonstrating the const type qualifier.
#include <iostream>
using std::cout;
using std::endl;

void tryToModifyArray( const int [] ); // function prototype

int main()
{
   int a[] = { 10, 20, 30 };

   tryToModifyArray( a );
   cout << a[ 0 ] << ' ' << a[ 1 ] << ' ' << a[ 2 ] << '\n';

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

```

Using **const** to prevent the function from modifying the array

Array **a** will be **const** when in the body of the function

```
// In function tryToModifyArray, "b" cannot be used
// to modify the original array "a" in main.
void tryToModifyArray( const int b[] )
{
   b[ 0 ] /= 2; // error
   b[ 1 ] /= 2; // error
   b[ 2 ] /= 2; // error
} // end function tryToModifyArray
```

Array cannot be modified; it is **const** within the body function

Borland C++ command-line compiler error message:

```
Error E2024 fig07_15.cpp 23: Cannot modify a const object
   in function tryToModifyArray(const int * const)
Error E2024 fig07_15.cpp 24: Cannot modify a const object
   in function tryToModifyArray(const int * const)
Error E2024 fig07_15.cpp 25: Cannot modify a const object
   in function tryToModifyArray(const int * const)
```

Microsoft Visual C++.NET compiler error message:

```
C:\cpphtp5_examples\ch07\fig07_15.cpp(23) : error C2166: l-value specifies
   const object
C:\cpphtp5_examples\ch07\fig07_15.cpp(24) : error C2166: l-value specifies
   const object
C:\cpphtp5_examples\ch07\fig07_15.cpp(25) : error C2166: l-value specifies
   const object
```

GNU C++ compiler error message:

```
fig07_15.cpp:23: error: assignment of read-only location
fig07_15.cpp:24: error: assignment of read-only location
fig07_15.cpp:25: error: assignment of read-only location
```

4-2使用类模板array创建数组

- 类模板array简介
- 使用类模板array实例化数组
- 基于范围的for语句
- 利用array对象存放成绩的GradeBook类

4.2.1 array模板类介绍

- array模板类

  模板类：定义一个通用的类，类中的数据成员是未指定的类型（抽象类型）

  类模板实例化：给模板类中的数据成员指定一种具体类型之后，就得到一个具体的类。

例如：array是一个数组类模板，具有所有数组的共同特性以及与数组相关的成员函数。但array中数据成员没有指定具体类型，当给其数据成员指定了类型后，形成了一个具体类型的数组类了。

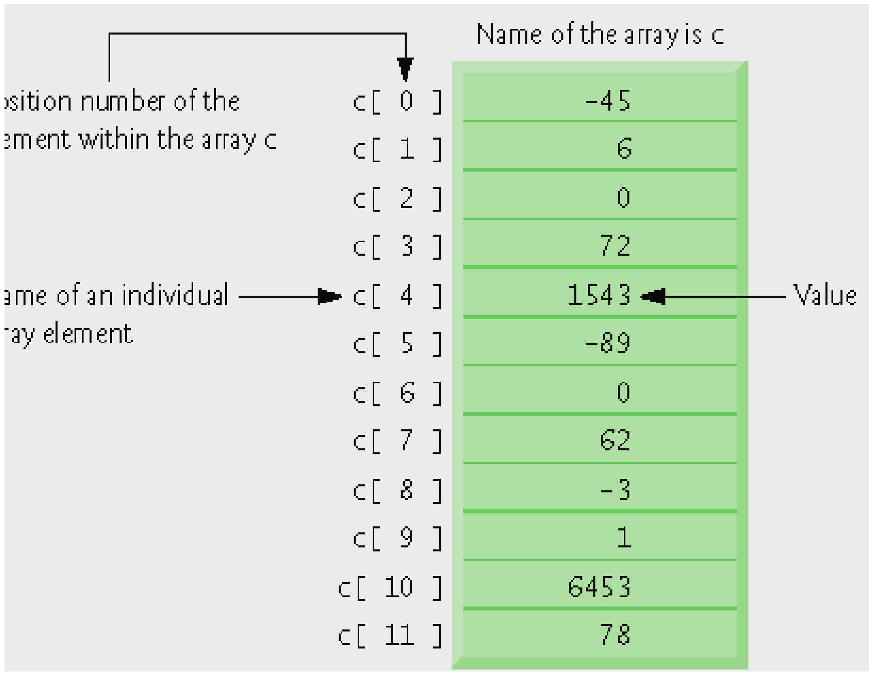
使用array类模板声明对象的方法：

Array<类型,大小> 对象名；

例如：

Array<int, 12> c;

//声明了一个包含12个int型元素的数组。



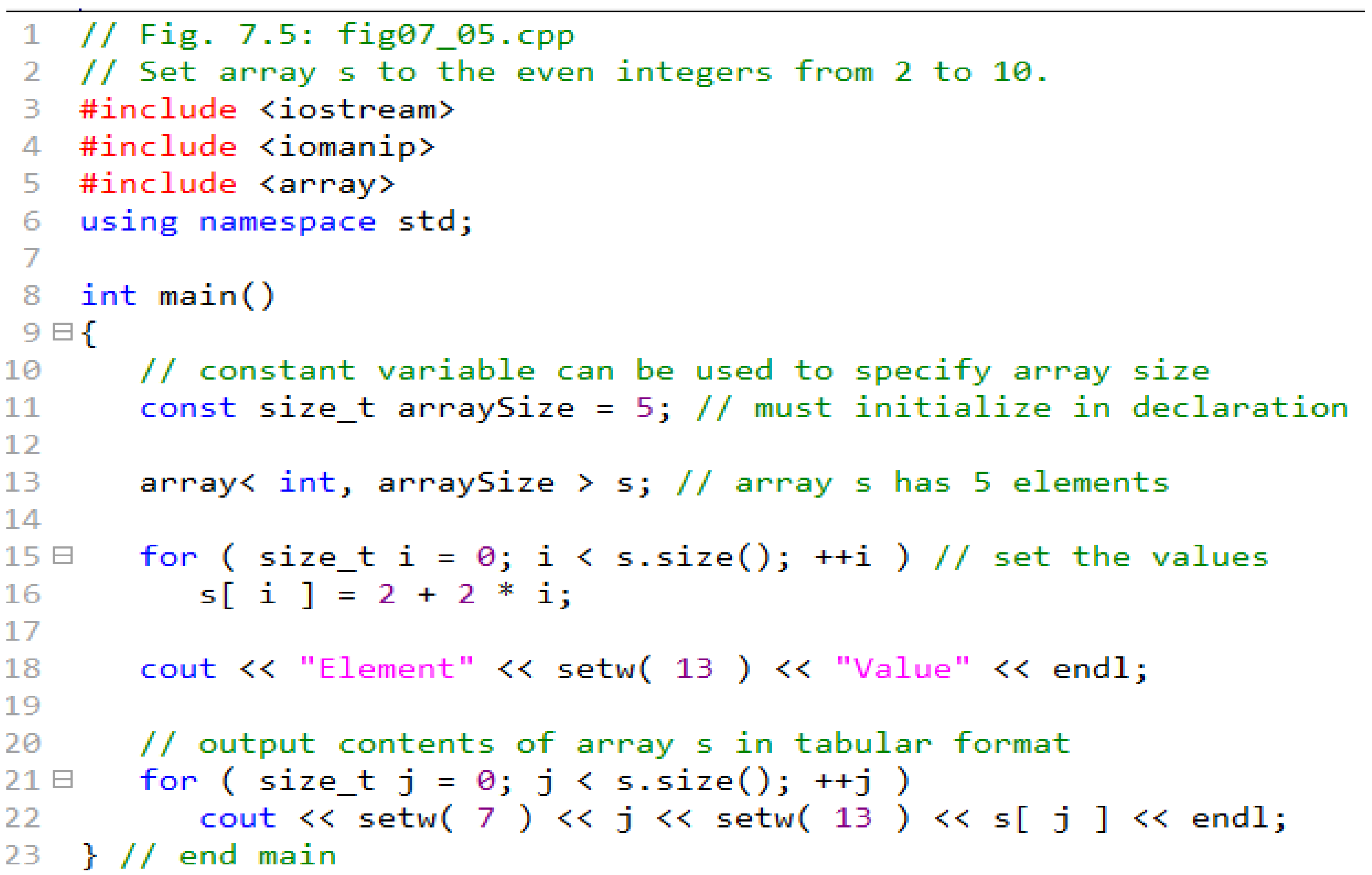
Array应用举例

```
#include <iostream>
#include <iomanip>              //使用setw（）等格式控制要将该头文件包含进来
#include <array>                //使用类模板array要将将该头文件包含进来
using namespace std;
int main()
{
  array< int, 5 > n;            // n is an array of 5 int values  n为包含5个整型元素的数组对象
  // initialize elements of array n to 0
  for ( size_t i = 0; i < n.size(); ++i )          //size_t在标准c++的std命名空间中被定义为无符号整形
    n[ i ] = 0;  // set element at location i to 0   //数组对象n的成员函数size()返回n中元素的个数
  cout << "Element" << setw( 13 ) << "Value" << endl;
  // output each array element's value
  for ( size_t j = 0; j < n.size(); ++j )
    cout << setw( 7 ) << j << setw( 13 ) << n[ j ] << endl;
} // end main
```

array< int, 5 > n = { 32, 27, 64, 18, 95 };

声明数组n的同时为其初始化。

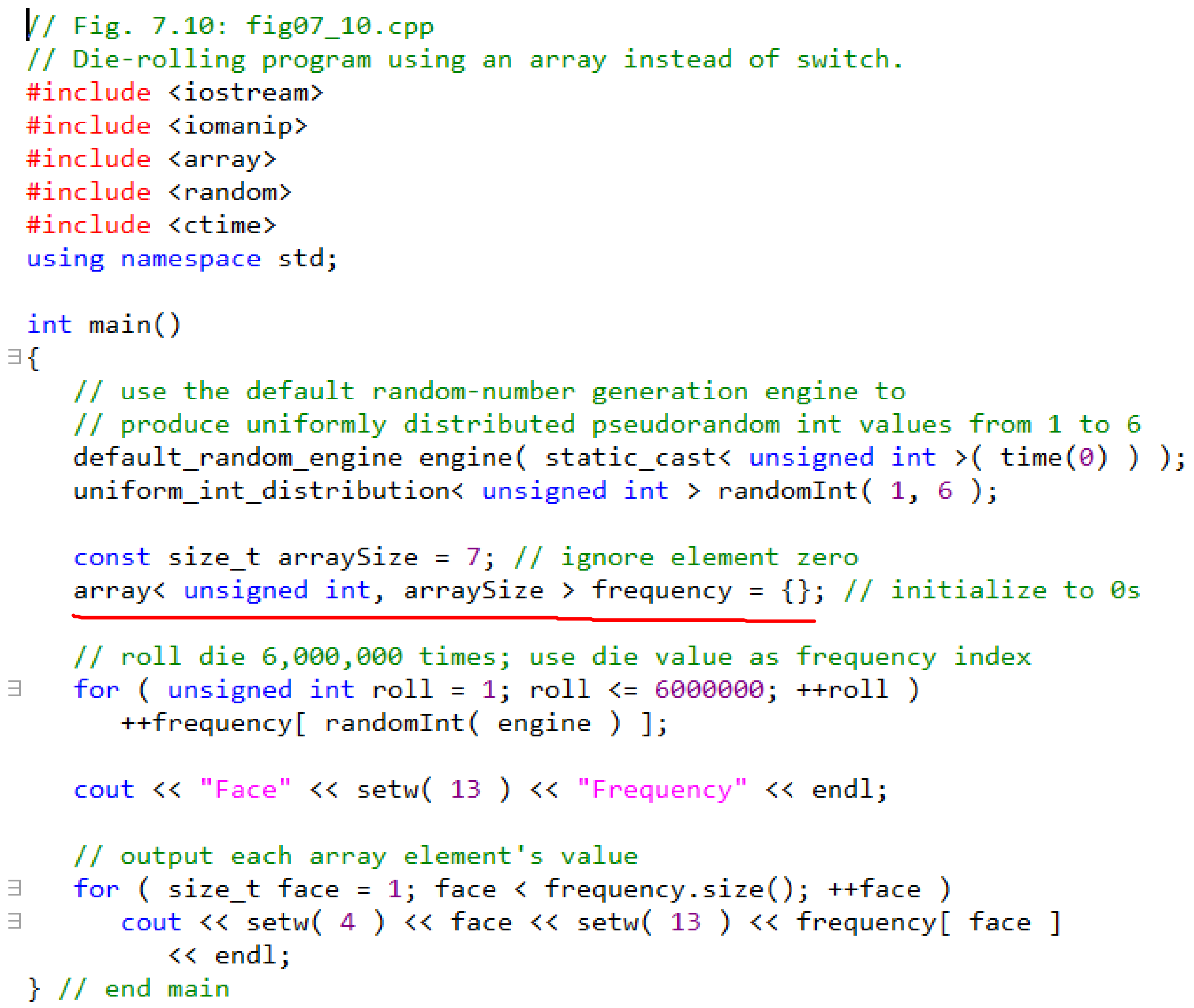
使用常变量指定array对象的大小



```cpp
// Fig. 7.5: fig07_05.cpp
// Set array s to the even integers from 2 to 10.
#include <iostream>
#include <iomanip>
#include <array>
using namespace std;

int main()
{
   // constant variable can be used to specify array size
   const size_t arraySize = 5; // must initialize in declaration

   array< int, arraySize > s; // array s has 5 elements

   for ( size_t i = 0; i < s.size(); ++i ) // set the values
      s[ i ] = 2 + 2 * i;

   cout << "Element" << setw( 13 ) << "Value" << endl;

   // output contents of array s in tabular format
   for ( size_t j = 0; j < s.size(); ++j )
      cout << setw( 7 ) << j << setw( 13 ) << s[ j ] << endl;
} // end main
```

关于随机数

- 所有在库中定义的随机数引擎都是伪随机数生成器，他们都利用了特定的算法实现，这些生成器都需要一个种子。种子可以是一个数值，或者是一个带有generate成员函数的对象。简单的应用中，用time作种子即可。
- 如果不设定种子，那么产生的随机数序列每次都一样，

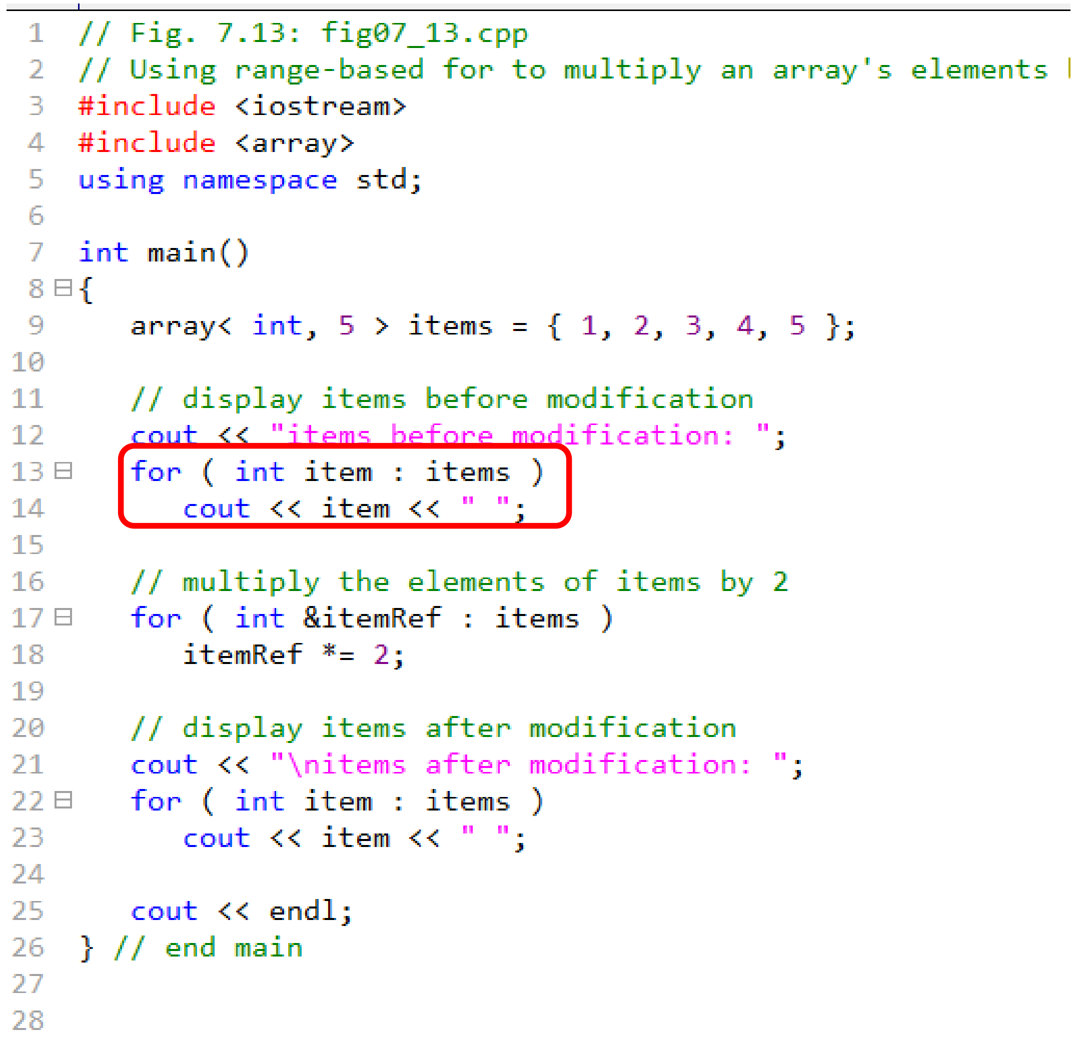


```cpp
// Fig. 7.10: fig07_10.cpp
// Die-rolling program using an array instead of switch.
#include <iostream>
#include <iomanip>
#include <array>
#include <random>
#include <ctime>
using namespace std;

int main()
{
   // use the default random-number generation engine to
   // produce uniformly distributed pseudorandom int values from 1 to 6
   default_random_engine engine( static_cast< unsigned int >( time(0) ) );
   uniform_int_distribution< unsigned int > randomInt( 1, 6 );

   const size_t arraySize = 7; // ignore element zero
   array< unsigned int, arraySize > frequency = {}; // initialize to 0s

   // roll die 6,000,000 times; use die value as frequency index
   for ( unsigned int roll = 1; roll <= 6000000; ++roll )
      ++frequency[ randomInt( engine ) ];

   cout << "Face" << setw( 13 ) << "Frequency" << endl;

   // output each array element's value
   for ( size_t face = 1; face < frequency.size(); ++face )
      cout << setw( 4 ) << face << setw( 13 ) << frequency[ face ]
         << endl;
} // end main
```

基于范围的for语句

语法:

For（范围变量：表达式）



```
// Fig. 7.13: fig07_13.cpp
// Using range-based for to multiply an array's elements
#include <iostream>
#include <array>
using namespace std;

int main()
{
   array< int, 5 > items = { 1, 2, 3, 4, 5 };

   // display items before modification
   cout << "items before modification: ";
   for ( int item : items )
      cout << item << " ";

   // multiply the elements of items by 2
   for ( int &itemRef : items )
      itemRef *= 2;

   // display items after modification
   cout << "\nitems after modification: ";
   for ( int item : items )
      cout << item << " ";

   cout << endl;
} // end main
```