Lecture 11: Array and Vector

Dr. Tsung-Wei Huang
Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, UT

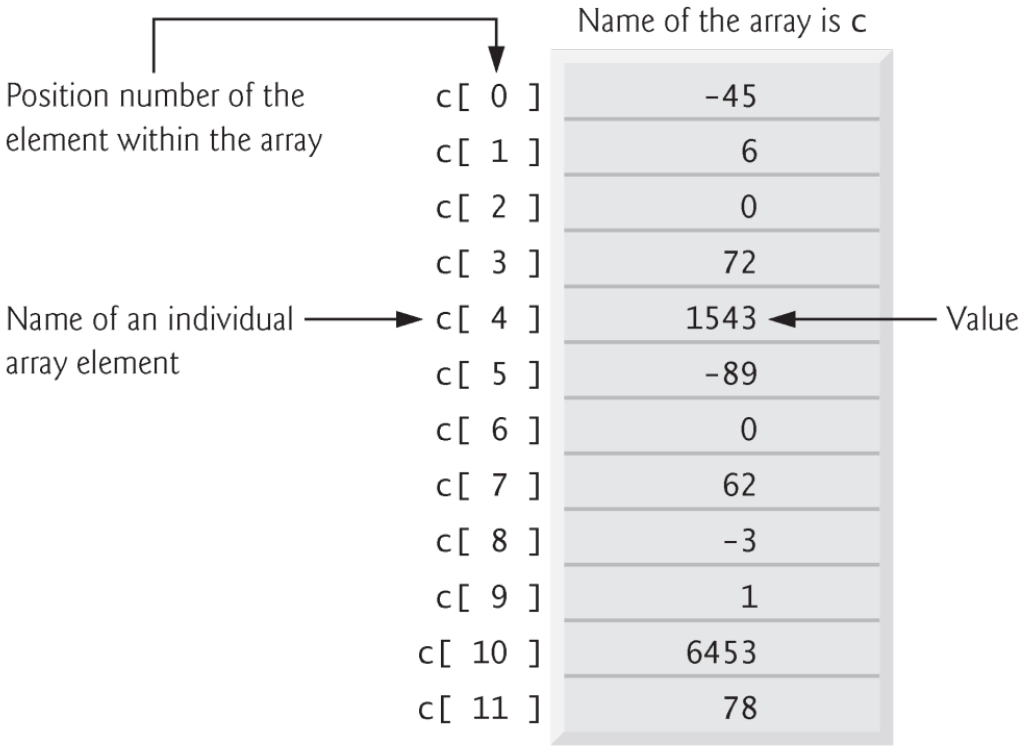
Learning Objective

- To use the array data structure to represent a set of related data items.
- To use arrays to store, sort and search lists and tables of values.
- To declare arrays, initialize arrays and refer to the individual elements of arrays.
- To pass arrays to functions.
- Basic searching and sorting techniques.
- To declare and manipulate multidimensional arrays.
- To use C++ Standard Library class template `vector`.
- To use C++ Standard Library class `string`.

Array

- **An array is a consecutive group of memory locations that all have the same type.**
- **To refer to a particular location or element in the array, specify the name of the array and the position number of that element.**
- **The position number is formally called a subscript or index.**
  - This number specifies the number of elements from the beginning of the array.
  - A subscript must be an integer or integer expression (using any integral type).
- **The first element in every array has subscript 0 (zero) and is sometimes called the zeroth element.**

Memory Layout of Array



Declaring an Array

- **Arrays occupy space in memory.**
- **To specify the type of the elements and the number of elements required by an array use a declaration of the form:**
  - *type arrayName[arraySize];*
  - Ex: `int n[5];` → 5 integer elements
- **The compiler reserves the appropriate amount of memory.**
  - All elements are put in a continuous memory space.

Array Example

```
1
2   // Initializing an array.
3   #include <iostream>
4   #include <iomanip>
5   using namespace std;
6
7   int main()
8   {
9      int n[ 10 ]; // n is an array of 10 integers
10
11     // initialize elements of array n to 0
12     for ( int i = 0; i < 10; i++ )
13        n[ i ] = 0; // set element at location i to 0
14
15     cout << "Element" << setw( 13 ) << "Value" << endl;
16
17     // output each array element's value
18     for ( int j = 0; j < 10; j++ )
19        cout << setw( 7 ) << j << setw( 13 ) << n[ j ] << endl;
20  } // end main
```

Array Example

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

Initializing an Array

- **The elements of an array also can be initialized by a brace-delimited comma-separated list of initializers.**
- **The previous program uses an initializer list to initialize an integer array with 10 values (line 10).**
- **If there are fewer initializers than elements in the array, the remaining array elements are initialized to zero.**
- **If the array size is omitted from a declaration with an initializer list, the compiler determines the array size automatically**
  - By counting the number of elements in the initializer list.
  - Ex: `int n[] = {1,2,3,4,5};` → 5 elements

Example: Initializing an Array

```
1
2  // Initializing an array in a declaration.
3  #include <iostream>
4  #include <iomanip>
5  using namespace std;
6
7  int main()
8  {
9     // use initializer list to initialize array n
10    int n[ 10 ] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
11
12    cout << "Element" << setw( 13 ) << "Value" << endl;
13
14    // output each array element's value
15    for ( int i = 0; i < 10; i++ )
16       cout << setw( 7 ) << i << setw( 13 ) << n[ i ] << endl;
17 } // end main
```

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

Specify the Array Size

```
1
2  // Set array s to the even integers from 2 to 20.
3  #include <iostream>
4  #include <iomanip>
5  using namespace std;
6
7  int main()
8  {
9     // constant variable can be used to specify array size
10    const int arraySize = 10;
11
12    int s[ arraySize ]; // array s has 10 elements
13
14    for ( int i = 0; i < arraySize; i++ ) // set the values
15       s[ i ] = 2 + 2 * i;
16
17    cout << "Element" << setw( 13 ) << "Value" << endl;
18
19    // output contents of array s in tabular format
20    for ( int j = 0; j < arraySize; j++ )
21       cout << setw( 7 ) << j << setw( 13 ) << s[ j ] << endl;
22 } // end main
```

const Qualifier

```
1
2  // A const variable must be initialized.
3
4  int main()
5  {
6     const int x; // Error: x must be initialized
7
8     x = 7; // Error: cannot modify a const variable
9  } // end main
```

Microsoft Visual C++ compiler error message:

```
C:\cpphtp6_examples\ch06\fig06_07.cpp(6) : error C2734: 'x' : const object
   must be initialized if not extern
C:\cpphtp6_examples\ch06\fig06_07.cpp(8) : error C3892: 'x' : you cannot
   assign to a variable that is const
```

GNU C++ compiler error message:

```
fig06_07.cpp:6: error: uninitialized const 'x'
fig06_07.cpp:8: error: assignment of read-only variable 'x'
```

Practice 1: Summing Elements in an Array

```
1
2  // Compute the sum of the elements of the array.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8     const int arraySize = 10; // constant variable indicating size of array
9     int a[ arraySize ] = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
10    int total = 0;
11
12    // sum contents of array a
13    for ( int i = 0; i < arraySize; i++ )
14       total += a[ i ];
15
16    cout << "Total of array elements: " << total << endl;
17 } // end main
```

```
Total of array elements: 849
```

Practice 2: Array as Counter

- Sometimes, programs use counter variables to summarize data, such as the results of a survey.
- In previous die-rolling program, we used separate counters to track the number of occurrences of each side of a die.

```
1
2   // Roll a six-sided die 6,000,000 times.
3   #include <iostream>
4   #include <iomanip>
5   #include <cstdlib> // contains function prototype for rand
6   using namespace std;
7
8   int main()
9   {
10     int frequency1 = 0; // count of 1s rolled
11     int frequency2 = 0; // count of 2s rolled
12     int frequency3 = 0; // count of 3s rolled
13     int frequency4 = 0; // count of 4s rolled
14     int frequency5 = 0; // count of 5s rolled
15     int frequency6 = 0; // count of 6s rolled
16
17     int face; // stores most recently rolled value
18
19     // summarize results of 6,000,000 rolls of a die
20     for ( int roll = 1; roll <= 6000000; roll++ )
21     {
22        face = 1 + rand() % 6; // random number from 1 to 6
23
```

```
24        // determine roll value 1-6 and increment appropriate counter
25        switch ( face )
26        {
27           case 1:
28              ++frequency1; // increment the 1s counter
29              break;
30           case 2:
31              ++frequency2; // increment the 2s counter
32              break;
33           case 3:
34              ++frequency3; // increment the 3s counter
35              break;
36           case 4:
37              ++frequency4; // increment the 4s counter
38              break;
39           case 5:
40              ++frequency5; // increment the 5s counter
41              break;
42           case 6:
43              ++frequency6; // increment the 6s counter
44              break;
45           default: // invalid value
46              cout << "Program should never get here!";
47        } // end switch
48     } // end for
```

C++ STL Vector

- **C-style pointer-based arrays have great potential for errors and are not flexible**
  - A program can easily "walk off" either end of an array, because C++ does not check whether subscripts fall outside the range of an array.
  - Two arrays cannot be meaningfully compared with equality operators or relational operators.
  - The size of the array must be passed as an additional argument when an array is passed to a general-purpose function.
  - One array cannot be assigned to another with the assignment operator(s).
- **C++ Standard class template vector represents a more robust type of array featuring many additional capabilities.**
  - Defined in header `<vector>` and belongs to namespace `std`.

Vector Example

```
1
2  // Demonstrating C++ Standard Library class template vector.
3  #include <iostream>
4  #include <iomanip>
5  #include <vector>
6  using namespace std;
7
8  void outputVector( const vector< int > & ); // display the vector
9  void inputVector( vector< int > & ); // input values into the vector
10
11 int main()
12 {
13    vector< int > integers1( 7 ); // 7-element vector< int >
14    vector< int > integers2( 10 ); // 10-element vector< int >
15
16    // print integers1 size and contents
17    cout << "Size of vector integers1 is " << integers1.size()
18       << "\nvector after initialization:" << endl;
19    outputVector( integers1 );
20
21    // print integers2 size and contents
22    cout << "\nSize of vector integers2 is " << integers2.size()
23       << "\nvector after initialization:" << endl;
24    outputVector( integers2 );
```

Vector Example

```
25
26      // input and print integers1 and integers2
27      cout << "\nEnter 17 integers:" << endl;
28      inputVector( integers1 );
29      inputVector( integers2 );
30
31      cout << "\nAfter input, the vectors contain:\n"
32         << "integers1:" << endl;
33      outputVector( integers1 );
34      cout << "integers2:" << endl;
35      outputVector( integers2 );
36
37      // use inequality (!=) operator with vector objects
38      cout << "\nEvaluating: integers1 != integers2" << endl;
39
40      if ( integers1 != integers2 )
41         cout << "integers1 and integers2 are not equal" << endl;
42
43      // create vector integers3 using integers1 as an
44      // initializer; print size and contents
45      vector< int > integers3( integers1 ); // copy constructor
46
```

Vector Example

```
47      cout << "\nSize of vector integers3 is " << integers3.size()
48         << "\nvector after initialization:" << endl;
49      outputVector( integers3 );
50
51      // use overloaded assignment (=) operator
52      cout << "\nAssigning integers2 to integers1:" << endl;
53      integers1 = integers2; // assign integers2 to integers1
54
55      cout << "integers1:" << endl;
56      outputVector( integers1 );
57      cout << "integers2:" << endl;
58      outputVector( integers2 );
59
60      // use equality (==) operator with vector objects
61      cout << "\nEvaluating: integers1 == integers2" << endl;
62
63      if ( integers1 == integers2 )
64         cout << "integers1 and integers2 are equal" << endl;
65
66      // use square brackets to create rvalue
67      cout << "\nintegers1[5] is " << integers1[ 5 ];
68
```

Vector Example

```
69       // use square brackets to create lvalue
70       cout << "\n\nAssigning 1000 to integers1[5]" << endl;
71       integers1[ 5 ] = 1000;
72       cout << "integers1:" << endl;
73       outputVector( integers1 );
74
75       // attempt to use out-of-range subscript
76       cout << "\nAttempt to assign 1000 to integers1.at( 15 )" << endl;
77       integers1.at( 15 ) = 1000; // ERROR: out of range
78    } // end main
79
80    // output vector contents
81    void outputVector( const vector< int > &array )
82    {
83       size_t i; // declare control variable
84
85       for ( i = 0; i < array.size(); i++ )
86       {
87          cout << setw( 12 ) << array[ i ];
88
89          if ( ( i + 1 ) % 4 == 0 ) // 4 numbers per row of output
90             cout << endl;
91       } // end for
92
```

Vector Example

```
93       if ( i % 4 != 0 )
94          cout << endl;
95    } // end function outputVector
96
97    // input vector contents
98    void inputVector( vector< int > &array )
99    {
100      for ( size_t i = 0; i < array.size(); i++ )
101         cin >> array[ i ];
102   } // end function inputVector
```

```
Size of vector integers1 is 7
vector after initialization:
          0          0          0          0
          0          0          0

Size of vector integers2 is 10
vector after initialization:
          0          0          0          0
          0          0          0          0
          0          0

Enter 17 integers:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
```

Vector Example

```
After input, the vectors contain:
integers1:
           1           2           3           4
           5           6           7
integers2:
           8           9          10          11
          12          13          14          15
          16          17

Evaluating: integers1 != integers2
integers1 and integers2 are not equal

Size of vector integers3 is 7
vector after initialization:
           1           2           3           4
           5           6           7

Assigning integers2 to integers1:
integers1:
           8           9          10          11
          12          13          14          15
          16          17
integers2:
           8           9          10          11
          12          13          14          15
          16          17
```

Vector Example

```
Evaluating: integers1 == integers2
integers1 and integers2 are equal

integers1[5] is 13

Assigning 1000 to integers1[5]
integers1:
           8           9          10          11
          12        1000          14          15
          16          17

Attempt to assign 1000 to integers1.at( 15 )
```

Platform specific error message will be dislayed

C++ STL Vector

- By default, all the elements of a `vector` object are set to `0`.
- `vector`s can be defined to store any data type.
- `vector` member function `size` obtain the number of elements in the `vector`.
- You can use square brackets, `[]`, to access the elements in a `vector`.
- `vector` objects can be compared with one another using the equality operators.
- You can create a new `vector` object that is initialized with a copy of an existing `vector`.

Search an Item in an Array

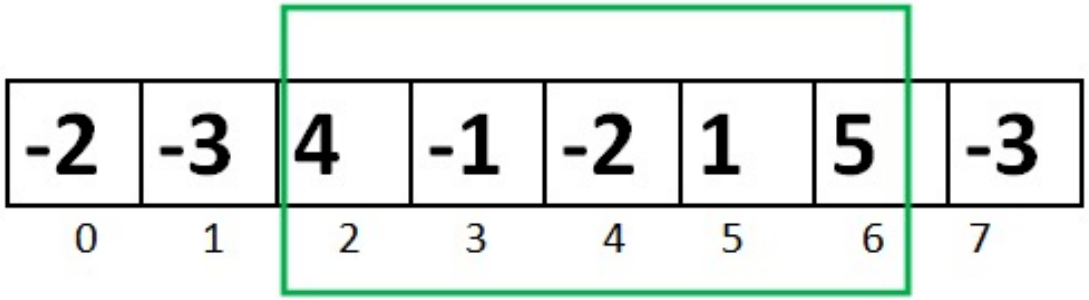
- **Often it may be necessary to determine whether an array contains a value that matches a certain key value.**
  - Called searching.
- **The linear search compares each element of an array with a search key.**
  - Because the array is not in any particular order, it's just as likely that the value will be found in the first element as the last.
  - On average, therefore, the program must compare the search key with half the elements of the array.
- **To determine that a value is not in the array, the program must compare the search key to every element of the array.**

Example 1

- **Input:**
  - A std::vector vec of integer numbers
  - An integer number v
- **Output**
  - “Yes”, if v can be found in vec
  - “No”, if v cannot be found in vec

Example 2: Maximum Subarray Sum Problem

- Given a sequence of integer number, find the largest sum of contiguous array numbers

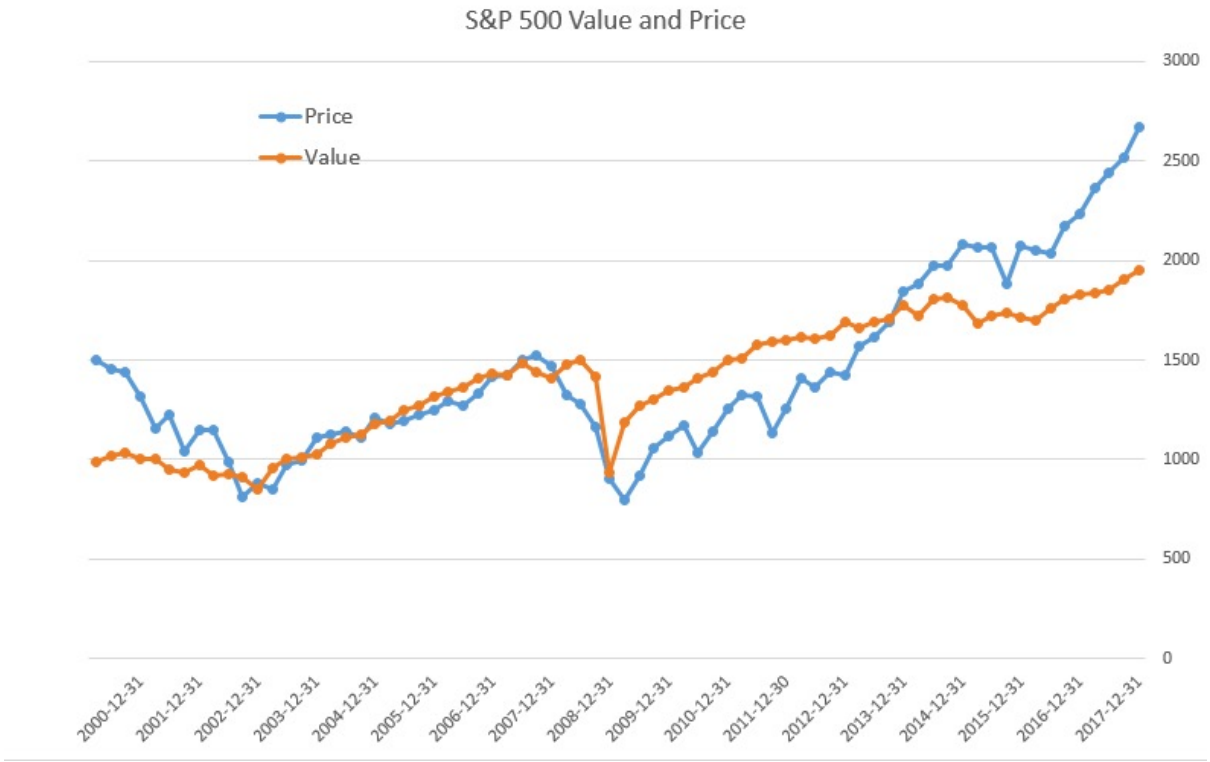


4 + (-1) + (-2) + 1 + 5 = 7

Maximum Contiguous Array Sum is 7

Practical Application

- A basic routine of financial computing



Maximum subarray sum to find the optimal long-term investment

Debrief

- The solution we presented has two-level loops
  - At worst, you need N^2 iterations, where N is the size of the input array

```
int brute_force(const std::vector<int>& D, int beg, int end) {
  int max = std::numeric_limits<int>::min();
  for (int i = beg; i < end; ++i) {
    int sum = 0;
    for (int j = i; j < end; ++j) {
      sum += D[j];
      max = std::max(sum, max);
    }
  }
  return max;
}
```

- Can we do better?

LAB

- **Write a program that takes an input *N* from users and then ask the user to type in *N* integers (N>=10), storing in a std::vector<int>; then, your program should output:**
  - The first three minimum numbers
  - The first three maximum numbers
- **Example: given {1, 4, 5, 9, 0, -1, 2, -100, -2}**
  - First three minimum: -100, -2, -1
  - First three maximum: 9, 5, 4