Chapter 7: Arrays

Hanoi University of Science and Technology

December 22, 2025

Slide contents are mainly based on the following documents:

- Tony Gaddis, Starting out with C++, Six Edition, Pearson, 2009
- Tony Gaddis, Power Points Slides:
  `https://www.umsl.edu/~antognolij/gaddis.html`

Using Parallel Arrays

- Parallel arrays: two or more arrays that contain related data
- A subscript is used to relate arrays: elements at same subscript are related
- Arrays may be of different types

Using Parallel Arrays

```
const int SIZE = 5; // Array size
int id[SIZE]; // student ID
double average[SIZE]; // course average
char grade[SIZE]; // course grade
...
for(int i = 0; i < SIZE; i++)
  cout << "Student ID: " << id[i]
       << " average:  " << average[i]
       << " grade:  " << grade[i]
       << endl;
```

Arrays as Function Arguments

- To pass an array to a function, just use the array name:
  ```
  showScores(tests);
  ```
- To define a function that takes an array parameter, use empty [] for array argument:
  ```
  void showScores(int []); // function prototype
  void showScores(int tests[]) // function header
  ```

Arrays as Function Arguments

- When passing an array to a function, it is common to pass array size so that function knows how many elements to process:

```
showScores(tests, ARRAY_SIZE);
```

- Array size must also be reflected in prototype, header:

```
void showScores(int [], int); // function prototype
void showScores(int tests[], int size) // header
```

Arrays as Function Arguments

Program 7-16

```
1  // This program demonstrates that an array element is passed
2  // to a function like any other variable.
3  #include <iostream>
4  using namespace std;
5
6  void showValue(int); // Function prototype
7
8  int main()
9  {
10     const int SIZE = 8;
11     int numbers[SIZE] = {5, 10, 15, 20, 25, 30, 35, 40};
12
13     for (int index = 0; index < SIZE; index++)
14         showValue(numbers[index]);
15     return 0;
16 }
17
18 //*********************************************
19 // Definition of function showValue.          *
20 // This function accepts an integer argument. *
21 // The value of the argument is displayed.    *
22 //*********************************************
23
24 void showValue(int num)
25 {
26     cout << num << " ";
27 }
```

Program Output

```
5 10 15 20 25 30 35 40
```

Arrays as Function Arguments

Program 7-17

```
1  // This program demonstrates an array being passed to a function.
2  #include <iostream>
3  using namespace std;
4
5  void showValues(int [], int); // Function prototype
6
7  int main()
8  {
9       const int ARRAY_SIZE = 8;
10      int numbers[ARRAY_SIZE] = {5, 10, 15, 20, 25, 30, 35, 40};
11
12      showValues(numbers, ARRAY_SIZE);
13      return 0;
14 }
15
16 //*************************************************
17 // Definition of function showValue.              *
18 // This function accepts an array of integers and  *
19 // the array's size as its arguments. The contents *
20 // of the array are displayed.                     *
21 //*************************************************
22
23 void showValues(int nums[], int size)
24 {
25      for (int index = 0; index < size; index++)
26          cout << nums[index] << " ";
27      cout << endl;
28 }
```

Program Output

```
5 10 15 20 25 30 35 40
```

Modifying Arrays in Functions

- Array names in functions are like reference variables – changes made to array in a function are reflected in actual array in calling function

Two-Dimensional Arrays

- Can define one array for multiple sets of data
- Like a table in a spreadsheet
- Use two size declarators in definition:

```
const int ROWS = 4, COLS = 3;
int exams[ROWS][COLS];
```

- First declarator is number of rows; second is number of columns

Two-Dimensional Arrays

- ```
  const int ROWS = 4, COLS = 3;
  int exams[ROWS][COLS];
  ```

exams[0][0]	exams[0][1]	exams[0][2]
exams[1][0]	exams[1][1]	exams[1][2]
exams[2][0]	exams[2][1]	exams[2][2]
exams[3][0]	exams[3][1]	exams[3][2]

- Use two subscripts to access element: `exams[2][2] = 86;`
  ```

2D Array Initialization

- Two-dimensional arrays are initialized row-by-row:

```
const int ROWS = 2, COLS = 2;
int exams[ROWS][COLS] = { {84, 78 }, {92, 97} };
```

Two-Dimensional Array as Parameter, Argument

- Use array name as argument in function call:

```
getExams(exams, 2);
```

- Use empty [] for row, size declarator for column in prototype, header:

```
const int COLS = 2;
// Prototype
void getExams(int [][COLS], int);
// Header
void getExams(int exams[][COLS], int rows)
```

Two-Dimensional Array as Parameter, Argument

Program 7-22

```
1  // This program demonstrates accepting a 2D array argument.
2  #include <iostream>
3  #include <iomanip>
4  using namespace std;
5
6  // Global constants
7  const int COLS = 4;       // Number of columns in each array
8  const int TBL1_ROWS = 3; // Number of rows in table1
9  const int TBL2_ROWS = 4; // Number of rows in table2
10
11 void showArray(const int [][COLS], int); // Function prototype
12
13 int main()
14 {
15      int table1[TBL1_ROWS][COLS] = {{1, 2, 3, 4},
16                                     {5, 6, 7, 8},
17                                     {9, 10, 11, 12}};
18      int table2[TBL2_ROWS][COLS] = {{10, 20, 30, 40},
19                                     {50, 60, 70, 80},
20                                     {90, 100, 110, 120},
21                                     {130, 140, 150, 160}};
```

Two-Dimensional Array as Parameter, Argument

```
22
23      cout << "The contents of table1 are:\n";
24      showArray(table1, TBL1_ROWS);
25      cout << "The contents of table2 are:\n";
26      showArray(table2, TBL2_ROWS);
27      return 0;
28  }
29
30  //*************************************************************
31  // Function Definition for showArray                          *
32  // The first argument is a two-dimensional int array with COLS *
33  // columns. The second argument, rows, specifies the number of *
34  // rows in the array. The function displays the array's contents. *
35  //*************************************************************
36
37  void showArray(const int numbers[][COLS], int rows)
38  {
39      for (int x = 0; x < rows; x++)
40      {
41          for (int y = 0; y < COLS; y++)
42          {
43              cout << setw(4) << numbers[x][y] << " ";
44          }
45          cout << endl;
46      }
47  }
```

Program Output

```
The contents of table1 are:
   1    2    3    4
   5    6    7    8
   9   10   11   12
The contents of table2 are:
  10   20   30   40
  50   60   70   80
  90  100  110  120
 130  140  150  160
```

Summing the Rows of a Two-Dimensional Array

- Given the following definitions:

```
const int NUM_STUDENTS = 3;
const int NUM_SCORES = 5;
double total; // Accumulator
double average; // To hold average scores
double scores[NUM_STUDENTS][NUM_SCORES] =
      {{88, 97, 79, 86, 94},
      {86, 91, 78, 79, 84},
      {82, 73, 77, 82, 89}};
```

Summing the Rows of a Two-Dimensional Array

```
// Get each student's average score.
for (int row = 0; row < NUM_STUDENTS; row++)
{
   // Set the accumulator.
   total = 0;
   // Sum a row.
   for (int col = 0; col < NUM_SCORES; col++)
      total += scores[row][col];
   // Get the average
   average = total / NUM_SCORES;
   // Display the average.
   cout << "Score average for student "
        << (row + 1) << " is " << average <<endl;
}
```

Summing the Columns of a Two-Dimensional Array

```
// Get the class average for each score.
for (int col = 0; col < NUM_SCORES; col++)
{
   // Reset the accumulator.
   total = 0;
   // Sum a column
   for (int row = 0; row < NUM_STUDENTS; row++)
      total += scores[row][col];
   // Get the average
   average = total / NUM_STUDENTS;
   // Display the class average.
   cout << "Class average for test " << (col + 1)
        << " is " << average << endl;
}
```

Arrays with Three or More Dimensions

- Can define arrays with any number of dimensions:

```
short rectSolid[2][3][5];
double timeGrid[3][4][3][4];
```

- When used as parameter, specify all but 1st dimension in prototype, heading:

```
void getRectSolid(short [][3][5]);
```

Introduction to `vector`

- A data type defined in the Standard Template Library
- Can hold values of any type:
  `vector<int> scores;`
- Automatically adds space as more is needed – no need to determine size at definition
- Can use [] to access elements

Declaring Vectors

- You must `#include<vector>`
- Declare a vector to hold int element:

```
vector<int> scores;
```

- Declare a vector with initial size 30:

```
vector<int> scores(30);
```

- Declare a vector and initialize all elements to 0:

```
vector<int> scores(30, 0);
```

Adding Elements to a Vector

- Use `push_back` member function to add element to a full array or to an array that had no defined size:

```
scores.push_back(75);
```

- Use size member function to determine size of a vector:

```
howbig = scores.size();
```

Removing Vector Elements

- Use `pop_back` member function to remove last element from vector:
  ```
  scores.pop_back();
  ```
- To remove all contents of vector, use clear member function:
  ```
  scores.clear();
  ```
- To determine if vector is empty, use empty member function:
  ```
  while (!scores.empty()) ...
  ```

More functions

Member Function	Description	Example
`at(elt)`	Returns the value of the element at position `elt` in the vector	`cout << vec1.at(i);`
`capacity()`	Returns the maximum number of elements a vector can store without allocating more memory	`maxelts = vec1.capacity();`
`reverse()`	Reverse the order of the elements in a vector	`vec1.reverse();`
`resize (elts,val)`	Add elements to a vector, optionally initializes them	`vec1.resize(5,0);`
`swap(vec2)`	Exchange the contents of two vectors	`vec1.swap(vec2);`

Exercises

- Challenging problems: 5, 8, 19 (1 + 2, 3 + 4, 5 + 6)
- Given a square matrix, print the elements on the main diagonal and the elements on the secondary diagonal.
- Given a square matrix, swap two columns.
- Given a square matrix, swap two columns.
- Given a M x N matrix. Find the value(s) that appear most frequently in the sequence.
- Given a vector of N numbers, sort these numbers in ascending order.
- Given a vector of N numbers, reverse the elements in the vector.
- Given a vector of N integers, print the set containing only distinct numbers (removing duplicates).