Advanced Programming

Multi-Dimensional Arrays in C++

Dr. Zeyad Safaa Younus Saffawi

Multi-Dimensional Arrays in C++

- In C++, arrays are not limited to one dimension. The language supports **multi-dimensional arrays**, which allow us to store data in the form of tables, matrices, or higher-dimensional structures.
- The general declaration form is:

type name [size1][size2]...[sizeN];

Each additional **pair** of **square brackets** represents another **dimension**.

Example of a **three-dimensional** array:

```
int x [5] [10] [4];
```

This declaration creates:

- 5 blocks
- Each block contains 10 rows
- Each row contains 4 columns

Two-Dimensional Arrays (2D Arrays)

- The most commonly used multidimensional array is the **two-dimensional array**.
- A two-dimensional array can be understood as: An array of arrays.
- It is often visualized as a **table** consisting of rows and columns.
- **Syntax:**

```
type array_name[x][y];
```

Where:

- x = number of rows
- y = number of columns

Example:

```
int a[3][4];
```

This creates:

- 3 rows
- 4 columns
- Total elements = 3 × 4 = 12

Conceptual Representation

- A 2D array can be visualized as:

	Col0	Col1	Col2	Col3
Row0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

- Each element is accessed using:

a[i][j]

- Where:
  - i → row index
  - j → column index
  - Indexing always starts from 0.

Initializing Two-Dimensional Arrays

- There are two main ways to initialize a 2D array.
- **Method 1: Row-wise Initialization**
- int a[3][4] = {

```
        {0, 1, 2, 3},       /* initializers for row indexed by 0 */
        {4, 5, 6, 7},       /* initializers for row indexed by 1 */
        {8, 9, 10, 11}      /* initializers for row indexed by 2 */
    };
```

  - Each inner brace represents one row.
- **Method 2: Flat Initialization**

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

  - The compiler fills elements row by row automatically.
- Both methods produce the same result.

Accessing 2D Array Elements

- To access an element, we must specify both indices:

```
int val = a[2][3];
```

This means:

- Row index = 2 (third row)
- Column index = 3 (fourth column)
- First index → row
- Second index → column

Processing 2D Arrays Using Nested Loops

Because a 2D array has rows and columns, we use **nested loops**.

Example:

```
#include <iostream>
using namespace std;
int main()
{
   int a[5][2] = {
      {0,0},
      {1,2},
      {2,4},
      {3,6},
      {4,8}
   };
   for(int i = 0; i < 5; i++)
   {
      for(int j = 0; j < 2; j++)
      {
         cout << "a[" << i << "][" << j << "] = ";  //Each element is printed with its position
         cout << a[i][j] << endl;
      }
   }
   return 0;
}
```

Output:

```
a[0][0] = 0
a[0][1] = 0
a[1][0] = 1
a[1][1] = 2
a[2][0] = 2
a[2][1] = 4
a[3][0] = 3
a[3][1] = 6
a[4][0] = 4
a[4][1] = 8
```

Memory Layout of 2D Arrays

- Although we think of 2D arrays as tables, they are stored in memory as a **continuous block**.
- C++ stores elements in **row-major order**, meaning:
  - Entire first row
  - Then entire second row
  - Then third row
- Understanding this is important for:
  - Performance
  - Pointer arithmetic
  - Advanced programming

Higher-Dimensional Arrays

- C++ allows arrays with more than two dimensions.
- Example:

```
int data[4][3][2];
```

- However, in practice:
  - Most applications use **1D** or **2D** arrays
  - Higher dimensions are less common

Practical Example

Write a C++ program to read a two-dimensional array of size 3 rows and 6 columns row by row.

```
#include <iostream>
using namespace std;
int main()
{
   int A[3][6];
   // Read array row by row
   for(int i = 0; i < 3; i++)
   {
      for(int j = 0; j < 6; j++)
      {
         cin >> A[i][j];
      }
   }
   return 0;
}
```

Practical Example

Write a C++ program that performs the following tasks:
1. Declare a two-dimensional integer array of size 3 × 6.
2. Read the elements of the array column by column.
3. Display the array elements row by row in matrix form.

```
#include <iostream>
using namespace std;
int main()
{
   int A[3][6];
   // Reading the array column by column
   for(int j = 0; j < 6; j++)
   {
      for(int i = 0; i < 3; i++)
      {
         cout << "Enter element A[" << i << "][" << j << "]: ";
         cin >> A[i][j];
      }
   }
   cout << "\nThe array elements are:\n";
   // Printing the array row by row
   for(int i = 0; i < 3; i++)
   {
      for(int j = 0; j < 6; j++)
      {
         cout << A[i][j] << " ";
      }
      cout << endl;
   }
   return 0;
}
```

Practical Example

```
#include <iostream>
using namespace std;
int main()
{
   int A[4][4];
   // Read the array
   cout << "Enter 16 elements:\n";
   for(int i = 0; i < 4; i++)
   {
      for(int j = 0; j < 4; j++)
      {
         cin >> A[i][j];
      }
   }
   // Print main diagonal elements
cout << "\nMain diagonal elements
are:\n";
for(int i = 0; i < 4; i++)
{
   for(int j = 0; j < 4; j++)
   {
      if(i == j)
         cout << A[i][j] << " ";
   }
}
}
```

Write a C++ program to:
Read a two-dimensional array of size **4 × 4**.
Print only the elements of the **main diagonal**.

Practical Example

```
#include <iostream>
using namespace std;
int main()
{
   int A[2][3];
   int B[6];
   int k = 0;
cout << "Enter 6 elements:\n"; // Read the 2D array
   for(int i = 0; i < 2; i++)
   {
      for(int j = 0; j < 3; j++)
      {
         cin >> A[i][j];
      }
   }
for(int i = 0; i < 2; i++) // Convert to 1D array (row by row)
   {
      for(int j = 0; j < 3; j++)
      {
         B[k] = A[i][j];
         k++;
      }
   }
cout << "\nOne-dimensional array:\n"; // Print the 1D array
   for(int i = 0; i < 6; i++)
   {
      cout << B[i] << " ";
   }
   return 0;
}
```

Write a C++ program to:

1.Read a two-dimensional array of size **2 × 3**.

2.Convert its elements into a one-dimensional array.

3.Print the one-dimensional array.

Practical Example

Write a C++ program to:
1.Read a square matrix of size **4 × 4**.
2.Print only the elements of the **secondary diagonal**.

```
#include <iostream>
using namespace std;
int main()
{
   int A[4][4];

   // Read the matrix
   cout << "Enter 16 elements:\n";
   for(int i = 0; i < 4; i++)
   {
      for(int j = 0; j < 4; j++)
      {
         cin >> A[i][j];
      }
   }
   // Print secondary diagonal
   cout << "\nSecondary diagonal elements are:\n";
   for(int i = 0; i < 4; i++)
   {
      cout << A[i][3 - i] << " ";
   }
   return 0;
}
```

Practical Example

Write a C++ program to:
1.Read a square matrix of size **5 × 5**.
2.Convert the values of the **main diagonal** and the **secondary diagonal** to zeros.
3.Print the updated matrix.

```
#include <iostream>
using namespace std;
int main()
{
   int A[5][5];
cout << "Enter 25 elements:\n";       // Read the matrix
   for(int i = 0; i < 5; i++)
   {
      for(int j = 0; j < 5; j++)
      {
         cin >> A[i][j];
      }
   }
for(int i = 0; i < 5; i++)       // Convert main and secondary diagonals to zero
   {
      for(int j = 0; j < 5; j++)
      {
         if(i == j || i + j == 4)
         {
            A[i][j] = 0;
         }
      }
   }
cout << "\nUpdated Matrix:\n";                // Print updated matrix
   for(int i = 0; i < 5; i++)
   {
      for(int j = 0; j < 5; j++)
      {
         cout << A[i][j] << " ";
      }
      cout << endl;
   }
   return 0;
}
```

References

- Gaddis, T. (2014). *Starting out with C++: From control structures through objects* (8th ed.). Pearson.
- Soulié, J. (2007, April 24). C++ language tutorial. cplusplus.com.
- Tutorials Point. (n.d.). *Learn C++ programming language*. Tutorials Point.