2D Arrays

Examples:

1. Accessing elements in a 2D Array
2. An uninitialized 2D Array
3. Partial Initialization of a 2D Array
4. Using incorrect indices in a 2D Array
5. Printing rows and columns in a 2D Array
6. Print the entire table
7. Square Tables
8. Copy 1D Array to a 2D Array

There are several problems that require array of more than one dimension. For example, if we want to work with a table that has both rows and columns, a 1D array would not suffice.

There are several problems that require array of more than one dimension. For example, if we want to work with a table that has both rows and columns, a 1D array would not suffice.

1. Box scores in baseball are reported with one player name listed for each row and one statistic listed for each column.

There are several problems that require array of more than one dimension. For example, if we want to work with a table that has both rows and columns, a 1D array would not suffice.

1. Box scores in baseball are reported with one player name listed for each row and one statistic listed for each column.

2. Another example is an instructor's grade book, in which a student name is listed for each row and his or her test/quiz/lab scores are listed for each column.

Note:
A 2D array is a collection of 1D arrays: each element in a 2D array is a 1D array

1: Accessing elements in a 2D array: 10

It is the programmer's responsibility to keep the indices within the range!

```
int table [6][6] =
{   //  0    1    2    3    4
         {10, 90, 50, 60, 20}, //  0
         {55, 15, 25, 45, 35}, //  1
         {19, 49, 89, 29, 59}, //  2
         {72, 82, 92, 52, 22}  //  3
};
```

// Code a statement to print the first element in the first row.

```
cout << table[0][0] << endl;
```

1: Accessing elements in a 2D array: 92

```
int table [6][6] =
{   //  0    1    2    3    4
        {10, 90, 50, 60, 20}, //  0
        {55, 15, 25, 45, 35}, //  1
        {19, 49, 89, 29, 59}, //  2
        {72, 82, 92, 52, 22}  //  3
};
```

// … the third element in the last row: 92

```
cout <<                   << endl;
```

1: Accessing elements in a 2D array: 92

```
int table [6][6] =
{    //  0    1    2    3    4
        {10, 90, 50, 60, 20}, //  0
        {55, 15, 25, 45, 35}, //  1
        {19, 49, 89, 29, 59}, //  2
        {72, 82, 92, 52, 22}  //  3
};
```

// … the third element in the last row: 92

```
cout <<  table[3][2] << endl;
```

1: Accessing elements in a 2D array: 15

```
int table [6][6] =
{   //  0    1    2    3    4
        {10, 90, 50, 60, 20}, //  0
        {55, 15, 25, 45, 35}, //  1
        {19, 49, 89, 29, 59}, //  2
        {72, 82, 92, 52, 22}  //  3
};
```

// … the second element in the second row: 15

```
cout <<                          << endl;
```

1: Accessing elements in a 2D array: 15

```
int table [6][6] =
{   //  0    1    2    3    4
        {10, 90, 50, 60, 20}, //  0
        {55, 15, 25, 45, 35}, //  1
        {19, 49, 89, 29, 59}, //  2
        {72, 82, 92, 52, 22}  //  3
};
```

// … the second element in the second row: 15

```
cout << table[1][1]  << endl;
```

1: Accessing elements in a 2D array: 49

```
int table [6][6] =
{   //  0    1    2    3    4
          {10, 90, 50, 60, 20}, //  0
          {55, 15, 25, 45, 35}, //  1
          {19, 49, 89, 29, 59}, //  2
          {72, 82, 92, 52, 22}  //  3
};
```

// … the second element in the third row: 49

```
cout <<                    << endl;
```

1: Accessing elements in a 2D array: 49

```
int table [6][6] =
{    //   0    1    2    3    4
        {10, 90, 50, 60, 20}, //  0
        {55, 15, 25, 45, 35}, //  1
        {19, 49, 89, 29, 59}, //  2
        {72, 82, 92, 52, 22}  //  3
};
```

// … the second element in the third row: 49

```
cout <<  table[2][1] << endl;
```

1: Accessing elements in a 2D array: 22

```
int table [6][6] =
{   //  0    1    2    3    4
        {10, 90, 50, 60, 20}, //  0
        {55, 15, 25, 45, 35}, //  1
        {19, 49, 89, 29, 59}, //  2
        {72, 82, 92, 52, 22}  //  3
};
```

// … the last element in the last row: 22

```
cout <<                 << endl;
```

1: Accessing elements in a 2D array: 22

```
int table [6][6] =
{   //  0    1    2    3    4
        {10, 90, 50, 60, 20}, //  0
        {55, 15, 25, 45, 35}, //  1
        {19, 49, 89, 29, 59}, //  2
        {72, 82, 92, 52, 22}  //  3
};
```

// … the last element in the last row: 22

```
cout << table[3][4]  << endl;
```

1: Accessing elements in a 2D array: 89

```
int table [6][6] =
{   //  0    1    2    3    4
        {10, 90, 50, 60, 20}, //  0
        {55, 15, 25, 45, 35}, //  1
        {19, 49, 89, 29, 59}, //  2
        {72, 82, 92, 52, 22}  //  3
};
```

// … the third element in the third row: 89

```
cout <<                << endl;
```

1: Accessing elements in a 2D array: 89

```
int table [6][6] =
{   //  0    1    2    3    4
       {10, 90, 50, 60, 20}, //  0
       {55, 15, 25, 45, 35}, //  1
       {19, 49, 89, 29, 59}, //  2
       {72, 82, 92, 52, 22}  //  3
};
```

// … the third element in the third row: 89

```
cout << table[2][2]  << endl;
```

1: Accessing elements in a 2D array:

```
int table [6][6] =
{   //  0    1    2    3    4
        {10, 90, 50, 60, 20}, //  0
        {55, 15, 25, 45, 35}, //  1
        {19, 49, 89, 29, 59}, //  2
        {72, 82, 92, 52, 22}  //  3
};
```

// Predict the output. Explain.

```
cout << table[0][5] << endl;
```

```
int table [6][6] =
{    //  0    1    2    3    4   5
         {10, 90, 50, 60, 20, 0},
         {55, 15, 25, 45, 35, 0},
         {19, 49, 89, 29, 59, 0},
         {72, 82, 92, 52, 22, 0},
         { 0,  0,  0,  0,  0, 0},
         { 0,  0,  0,  0,  0, 0},

};
```

// Predict the output. Explain.

```
cout << table[0][5] << endl; // 0
```

Note:
This array holds 6 x 6 = 36 integers. Since the array is partially initialized, all uninitialized elements are set to 0!

```
    int table [6][6] =
    {   //  0    1    2    3    4   5
           {10, 90, 50, 60, 20, 0}, //  0
           {55, 15, 25, 45, 35, 0}, //  1
           {19, 49, 89, 29, 59, 0}, //  2
           {72, 82, 92, 52, 22, 0}, //  3
           { 0,  0,  0,  0,  0, 0}, //  4
           { 0,  0,  0,  0,  0, 0}, //  5

    };
```

// Predict the output. Explain.

```
cout << table[0][6] << endl;
```

```
int table [6][6] =
{   //  0    1    2    3    4   5
         {10, 90, 50, 60, 20, 0}, //  0
         {55, 15, 25, 45, 35, 0}, //  1
         {19, 49, 89, 29, 59, 0}, //  2
         {72, 82, 92, 52, 22, 0}, //  3
         { 0,  0,  0,  0,  0, 0}, //  4
         { 0,  0,  0,  0,  0, 0}, //  5
};
```

// Predict the output. Explain.

```
cout << table[0][6] << endl; // 55
```

Note:
Index 6 is an invalid index for a column! No errors are reported, instead the next element in the memory will be accessed, and since the array is stored row by row, this will be the first element in the next row.

2. Accessing elements in a 2D array. Explain the output.

```
int table_one[2][3];

cout << table_one[0][0] << endl;
cout << table_one[0][1] << endl;
cout << table_one[0][2] << endl;

cout << table_one[1][0] << endl;
cout << table_one[1][1] << endl;
cout << table_one[1][2] << endl;
```

```
2686760
4273158
4273056
7095488
68
2130567168
```

Note:
This 2D array has been declared but not initialized, therefore it has junk values.

3. Accessing elements in a 2D array. Explain the output.

```
int table_two[3][5] =
{
      {10, 90, 50},
      {55, 15, 25}
};
```

```
  10   90   50

  55   15   25

   0    0    0
```

```
cout <<   table_two[0][0] << "  ";
cout <<   table_two[0][1] << "  ";
cout <<   table_two[0][2] << endl;

cout <<   table_two[1][0] << "  ";
cout <<   table_two[1][1] << "  ";
cout <<   table_two[1][2] << endl;

cout <<   table_two[0][3] << "  ";
cout <<   table_two[1][3] << "  ";
cout <<   table_two[2][0] << endl;
```

3. Accessing elements in a 2D array. Explain the output.

```
int table_two[3][5] =
{
      {10, 90, 50, 0, 0},
      {55, 15, 25, 0, 0}
      { 0,  0,  0, 0, 0}
};
```

```
  10   90   50

  55   15   25

   0    0    0
```

```
{ {10, 90, 50, 0, 0}, {55, 15, 25, 0, 0}, {0, 0, 0, 0, 0} };
```

4. Accessing elements in a 2D array. Explain the output.

```
int table_two[2][3] =
{
      {10, 90, 50},
      {55, 15, 25}
};
```

```
   55   15   25
```

```
cout << setw(5) << table_three[0][3];
cout << setw(5) << table_three[0][4];
cout << setw(5) << table_three[0][5] << endl;
```

```
{ {10, 90, 50}, {55, 15, 25} };
    0   1   2      3   4   5
```

4. Accessing elements in a 2D array. Explain the output.

```
int table_two[2][3] =
{
      {10, 90, 50},
      {55, 15, 25}
};
```

```
    55     15     25
```

```
cout <<   table_three[0][3] << "  ";
cout <<   table_three[0][4] << "  ";
cout <<   table_three[0][5] << endl;
```

```
{ {10, 90, 50}, {55, 15, 25} };
   0    1   2      3    4   5
```

5. Printing rows and columns

```
int table [ROWS] [COLS] =
{
    {10, 90, 50, 60, 20},
    {55, 15, 25, 45, 35},
    {19, 49, 89, 29, 59},
    {72, 82, 92, 52, 22}
};
int rows = 4;
int cols = 5;
int r, c;

//  Print the first row
for (        ;            ;      )
    cout << table[ ][ ] << " ";
cout << endl;
```

5. Printing rows and columns

```
int table [ROWS] [COLS] =
{ // 0    1    2    3    4
    {10, 90, 50, 60, 20}, // 0
    {55, 15, 25, 45, 35},
    {19, 49, 89, 29, 59},
    {72, 82, 92, 52, 22}
};
int rows = 4;
int cols = 5;
int r, c;
```

```
//  Print the first row
for ( c = 0 ; c < cols ; c++ )
     cout << table[0][c] << " ";
cout << endl;
```

5. Printing rows and columns

```
int table [ROWS] [COLS] =
{
    {10, 90, 50, 60, 20},
    {55, 15, 25, 45, 35},
    {19, 49, 89, 29, 59},
    {72, 82, 92, 52, 22}
};
int rows = 4;
int cols = 5;
int r, c;

//  Print the first column
for (        ;           ;       )
     cout << table[ ][ ] << endl;
```

5. Printing rows and columns

```
int table [ROWS] [COLS] =
{ // 0
    {10, 90, 50, 60, 20}, // 0
    {55, 15, 25, 45, 35}, // 1
    {19, 49, 89, 29, 59}, // 2
    {72, 82, 92, 52, 22}  // 3
};
int rows = 4;
int cols = 5;
int r, c;

// Print the first column
for ( r = 0 ; r < rows ; r++ )
    cout << table[r][0] << endl;
```

5. Printing rows and columns

```
int table [ROWS] [COLS] =
{
    {10, 90, 50, 60, 20},
    {55, 15, 25, 45, 35},
    {19, 49, 89, 29, 59},
    {72, 82, 92, 52, 22}
};
int rows = 4;
int cols = 5;
int r, c;

// Print the third row
for (        ;            ;      )
     cout << table[ ][ ] << " ";
cout << endl;
```

5. Printing rows and columns

```
int table [ROWS] [COLS] =
{ // 0    1    2    3    4
    {10, 90, 50, 60, 20},
    {55, 15, 25, 45, 35},
    {19, 49, 89, 29, 59}, // 2
    {72, 82, 92, 52, 22}
};
int rows = 4;
int cols = 5;
int r, c;

//  Print the third row
for ( c = 0 ; c < cols ; c++ )
     cout << table[2][c] << " ";
cout << endl;
```

5. Printing rows and columns

```
int table [ROWS] [COLS] =
{
    {10, 90, 50, 60, 20},
    {55, 15, 25, 45, 35},
    {19, 49, 89, 29, 59},
    {72, 82, 92, 52, 22}
};
int rows = 4;
int cols = 5;
int r, c;

//  Print the last row
for (        ;            ;       )
     cout << table[ ][ ] << " ";
cout << endl;
```

5. Printing rows and columns

```
int table [ROWS] [COLS] =
{ // 0    1    2    3    4
    {10, 90, 50, 60, 20},
    {55, 15, 25, 45, 35},
    {19, 49, 89, 29, 59},
    {72, 82, 92, 52, 22} // 3
};
int rows = 4;
int cols = 5;
int r, c;

//  Print the last row
for ( c = 0 ; c < cols ; c++ )
     cout << table[3][c] << " ";
cout << endl;
```

5. Printing rows and columns

```
int table [ROWS] [COLS] =
{
    {10, 90, 50, 60, 20},
    {55, 15, 25, 45, 35},
    {19, 49, 89, 29, 59},
    {72, 82, 92, 52, 22}
};
int rows = 4;
int cols = 5;
int r, c;

//  Print the second column
for (        ;           ;      )
     cout << table[ ][ ] << endl;
```

5. Printing rows and columns

```
int table [ROWS] [COLS] =
{       // 1
    {10, 90, 50, 60, 20}, // 0
    {55, 15, 25, 45, 35}, // 1
    {19, 49, 89, 29, 59}, // 2
    {72, 82, 92, 52, 22}  // 3
};
int rows = 4;
int cols = 5;
int r, c;

//  Print the second column
for ( r = 0 ; r < rows ; r++ )
    cout << table[r][1] << endl;
```

5. Printing rows and columns

```
int table [ROWS] [COLS] =
{
    {10, 90, 50, 60, 20},
    {55, 15, 25, 45, 35},
    {19, 49, 89, 29, 59},
    {72, 82, 92, 52, 22}
};
int rows = 4;
int cols = 5;
int r, c;

//  Print the last column
for (        ;            ;      )
     cout << table[ ][ ] << endl;
```

5. Printing rows and columns

```
int table [ROWS] [COLS] =
{                          // 4
    {10, 90, 50, 60, 20}, // 0
    {55, 15, 25, 45, 35}, // 1
    {19, 49, 89, 29, 59}, // 2
    {72, 82, 92, 52, 22}  // 3
};
int rows = 4;
int cols = 5;
int r, c;

//  Print the last column
for ( r = 0 ; r < rows ; r++ )
     cout << table[r][4] << endl;
```

6. Printing the entire table using different formats

```
int table [ROWS] [COLS] =
{
   {10, 90, 50, 60, 20},
   {55, 15, 25, 45, 35},
   {19, 49, 89, 29, 59},
   {72, 82, 92, 52, 22}
};
int rows = 4;
int cols = 5;
```

// *Print the entire table one row at a time*

6. Printing the entire table using different formats

```
int table [ROWS] [COLS] =
{
    {10, 90, 50, 60, 20},
    {55, 15, 25, 45, 35},
    {19, 49, 89, 29, 59},
    {72, 82, 92, 52, 22}
};
int rows = 4;
int cols = 5;
```

```
OUTPUT

 10   90   50   60   20
 55   15   25   45   35
 19   49   89   29   59
 72   82   92   52   22
```

```
// Print the entire table one row at a time
for ( int r = 0 ; r < rows ; r++ )
{
    for ( int c = 0 ; c < cols ; c++ )
        cout << table[r][c] << " ";
    cout << endl;
}
```

6. Printing the entire table using different formats

```
int table [ROWS] [COLS] =
{
    {10, 90, 50, 60, 20},
    {55, 15, 25, 45, 35},
    {19, 49, 89, 29, 59},
    {72, 82, 92, 52, 22}
};
int rows = 4;
int cols = 5;


// Print the entire table one column at a time
```

6. Printing the entire table using different formats

```
int table [ROWS] [COLS] =
{
    {10, 90, 50, 60, 20},
    {55, 15, 25, 45, 35},
    {19, 49, 89, 29, 59},
    {72, 82, 92, 52, 22}
};
int rows = 4;
int cols = 5;
```

```
OUTPUT

  10  55  19  72

  90  15  49  82

  50  25  89  92

  60  45  29  52

  20  35  59  22
```

```
// Print the entire table one column at a time
for ( int c = 0 ; c < cols ; c++ )
{
    for ( int r = 0 ; r < rows ; r++ )
        cout << table[r][c] << " ";
    cout << endl;
}
```

7. Square Tables (same number of rows and columns)

-3	15	15	15	15
99	-3	15	15	15
99	99	-3	15	15
99	99	99	-3	15
99	99	99	99	-3

Exercise: Write a code fragment that assigns:
-3 to the elements located on the left-right diagonal,
99 to the elements below the left-right diagonal, and
15 to the elements above the left-right diagonal

7. The following program fills the diagonal of a square array with **-3**, the lower left triangle with **99**, and the upper right triangle with **15**. Rewrite it without using if statement at all (OK more loops). What approach do you think it is better? Defend your answer.

```
for (r = 0; r < size; r++)
{
   for (c = 0; c < size; c++)
   {
      if (r == c)
         table[r][c] = -3;
      else
         if (r > c)
            table[r][c] = 99;
         else
            table[r][c] = 15;
   }
}
```

	0	1	2	3	4
0	-3	15	15	15	15
1	99	-3	15	15	15
2	99	99	-3	15	15
3	99	99	99	-3	15
4	99	99	99	99	-3

1st approach: using if statements

	0	1	2	3	4
0	-3	15	15	15	15
1	99	-3	15	15	15
2	99	99	-3	15	15
3	99	99	99	-3	15
4	99	99	99	99	-3

```
for (r = 0; r < size; r++)
{
    for (c = 0; c < r; c++)
        table[r][c] = 99;

    table[r][r] = -3;

    for (c = r + 1; c < size; c++)
        table[r][c] = 15;
}
```

2nd approach: without if statements, but more loops

```
for (r = 0; r < size; r++)
{
    table[r][r] = -3;
    for (c = 0; c < r; c++)
    {
        table[r][c] = 99;
        table[c][r] = 15;
    }
}
```

	0	1	2	3	4
0	-3	15	15	15	15
1	99	-3	15	15	15
2	99	99	-3	15	15
3	99	99	99	-3	15
4	99	99	99	99	-3

3rd approach: using only 2 loops

8. Write a program fragment that copies a one-dimensional array to a table.

10	90	15	70	25	99

10	90	15
70	25	99

8. Write a program fragment that copies a one-dimensional array to a table.

10	90	15	70	25	99

10	90	15
70	25	99

<u>Assumptions:</u>

list – the 1D array

  n – its number of elements

table – the 2D array

  rows – its number of rows

  cols – its number of columns

8. Write a program fragment that copies a one-dimensional array to a table.

10	90	15	70	25	99

10	90	15
70	25	99

```
i = 0;
for (r = 0; r < rows; r++)
{
    for (c = 0; c < cols; c++)
    {
        table[r][c] = list[i];
        i++;
    }
}
```

1st approach: using 2 loops

8. Write a program fragment that copies a one-dimensional array to a table.

10	90	15	70	25	99

10	90	15
70	25	99

```
for (i = 0; i < n; i++)
    table[i/cols][i%cols] = list[i];
```

2nd approach: using 1 loop

8. Write a program fragment that copies a one-dimensional array to a table.

10	90	15	70	25	99

10	90	15
70	25	99

```
if ( n == rows * cols )
    for (i = 0; i < n; i++)
        table[i/cols][i%cols] = list[i];
else
    cout << "Error!";
```

2nd approach: using 1 loop + validation!

2D Arrays

Examples:

- ✓ 1. Accessing elements in a 2D Array
- ✓ 2. An uninitialized 2D Array
- ✓ 3. Partial Initialization of a 2D Array
- ✓ 4. Using incorrect indices in a 2D Array
- ✓ 5. Printing rows and columns in a 2D Array
- ✓ 6. Print the entire table
- ✓ 7. Square Tables
- ✓ 8. Copy 1D Array to a 2D Array