# 2D Arrays and Functions

Examples:

1. Passing one element of a 2D Array by value
2. Passing one element of a 2D Array by reference
3. Passing the entire table
4. Passing one row of a 2D Array
5. Passing the index of a column
6. Fahrenheit to Celsius
7. Class average for each quiz
8. Highest quiz for each students
9. A maze
10. Read data from file into a 2D Array

```
const int MAX_ROWS = 25;
const int MAX_COLS = 50;

void printElement(int  num);

int main( void )
{
      int table[MAX_ROWS][MAX_COLS];
      int rows;
      int cols;

      // code to read data
      printElement(table[0][0]);
      return 0;
}
```

Fill in the blanks according to the requirements.

*1. Passing one element of a 2D array.*

*Call **printElement**: pass the last element on the second row: 35*

```
printElement( ?????????????? );
```

```
void printElement( int  num )
{
     cout << num << endl;
     num = 99;
}
```

```
10   15   20
25   30   35
40   45   50
55   60   65

35

10   15   20
25   30   35
40   45   50
55   60   65
```

*1. Passing one element of a 2D array.*

```
printElement( table[1][2] );
```

```
void printElement( int  num )
{
     cout << num << endl;
     num = 99;
}
```

```
10   15   20
25   30   35
40   45   50
55   60   65

35

10   15   20
25   30   35
40   45   50
55   60   65
```

*2. Passing an element of a 2D array by reference.*

```
changeElement ( ???????????? );
```

| | | |
|---|---|---|
| 10 | 15 | 20 |
| 25 | 30 | 99 |
| 40 | 45 | 50 |
| 55 | 60 | 65 |

```
void changeElement( int &num )
{
     num = 99;
}
```

*2. Passing an element of a 2D array by reference.*

```
changeElement ( table[1][2] );
```

| | | |
|---|---|---|
| 10 | 15 | 20 |
| 25 | 30 | 99 |
| 40 | 45 | 50 |
| 55 | 60 | 65 |

```
void changeElement( int &num )
{
     num = 99;
}
```

*3. Passing the entire table (not really: the function does not receive a copy of the entire table; it receives a reference to the original table!)*

```
printTable( ?????????????? );
```

```
10   15   20
25   30  999
40   45   50
55   60   65
```

```
void printTable( int table[][MAX_COLS],
                 int  rows,
                 int  cols)
{     int r;
      int c;

      for (r = 0; r <  rows; r++)
      {      // Print the current row
             for(c = 0; c <  cols; c++)
                    cout << table[r][c];
             cout << endl ;
      }
      cout << endl;
}
```

*3. Passing the entire table (not really: the function does not receive a copy of the entire table; it receives a reference to the original table!)*

```
printTable( table,  rows,  cols );
```

```
10  15  20
25  30 999
40  45  50
55  60  65
```

```
void printTable( int table[][MAX_COLS],
                 int  rows,
                 int  cols)
{     int r;
      int c;

      for (r = 0; r <  rows; r++)
      {      // Print the current row
             for(c = 0; c <  cols; c++)
                    cout << table[r][c];
             cout << endl ;
      }
      cout << endl;
}
```

*Question: Why do we have to write MAX_COLS in the function definition?*

```
printTable( table,  rows,  cols );
```

```
10  15  20
25  30 999
40  45  50
55  60  65
```

```
void printTable( int table[][MAX_COLS],
                 int  rows,
                 int  cols)
{     int r;
      int c;

      for (r = 0; r <  rows; r++)
      {      // Print the current row
             for(c = 0; c <  cols; c++)
                   cout << table[r][c];
             cout << endl ;
      }
      cout << endl;
}
```

*Question: Why do we have to write MAX_COLS in the function definition?*
*Answer: It is an error to leave out MAX_COLS in a prototype declaration or a function definition. A 2D array is stored row by row. Each time we write table[r][c], a formula is used to calculate the memory location (address) of this element: r * MAX_COLS + c*

```
void printTable( int table[][MAX_COLS],
                 int  rows,
                 int  cols)
{       int r;
        int c;

        for (r = 0; r <  rows; r++)
        {       // Print the current row
                for(c = 0; c <  cols; c++)
                        cout << table[r][c];
                cout << endl ;
        }
        cout << endl;
}
```

*Let's consider the following table (2 rows, 4 columns):*

```
90 80 70 60 // row 0
10 20 30 40 // row 1
```

It is stored row by row:

```
90 80 70 60 10 20 30 40
```

Element 30 is in row 1, column 2:

```
table[1][2]
r = 1
c = 2
```

At what location is it stored in the 1D array (memory)?

```
r * MAX_COLS + c
1 * 4        + 2 => 6
```

*4. Passing a row*

```
printRow( ???????????????? );
```

```
10  15  20
25  30 999
40  45  50
55  60  65
```

```
void printRow (int  row[], int size )
{
      for (int c = 0; c < size; c++)
            cout << row[c] << " ";
      cout << endl;
}
```

# *4. Passing a row*

*Note:*

*It is possible to refer to a row in a 2D array with a single index since the table is stored row by row, therefore elements in any row are next to each other:* `10 15 20 25 30 999 40 45 50 55 60 65`

```
printRow( table[2], cols );
```

| | | |
|---|---|---|
| 10 | 15 | 20 |
| 25 | 30 | 999 |
| 40 | 45 | 50 |
| 55 | 60 | 65 |

```
void printRow (int  list[], int size )
{
      for (int c = 0; c < size; c++)
            cout << list[c] << " ";
      cout << endl;
}
```

## *5. Passing the index of a column*

*Note:*
*A 2D array is a collection of rows (not columns):*
*each row in a 2D array is a 1D array )*

```
printColumn( ????????????? );
```

```
10   15   20
25   30  999
40   45   50
55   60   65
```

```
void printColumn(int table[][MAX_COLS],
                 int theCol,
                 int rows)
{
    for( int r = 0; r <  rows; r++ )
        cout << table[r][theCol] << " ";
    cout << endl;
}
```

# *Passing the index of a column*

*Note:*
*It is not possible to refer to a column in a 2D array with a single index since the table is stored row by row, therefore elements in a column are NOT next to each other:* `10 15 20 25 30 999 40 45 50 55 60 65`

```
printColumn( table, 2, rows );
```

```
void printColumn(int table[][MAX_COLS],
                 int theCol,
                 int rows)
{
    for( int r = 0; r <  rows; r++ )
        cout << table[r][theCol] << " ";
    cout << endl;
}
```

```
10  15  20
25  30 999
40  45  50
55  60  65
```

**6.** Write a function that takes a table of Fahrenheit temperatures in and displays it as Celsius temperatures. Call it from main.

```
cel = ( fahr – 32 ) * 5 / 9
```

```
double tempTable [YEARS][MONTHS] =
{
   {48.29, 53.00, 55.97}, //  ... more values here: a total of 12
   {50.35, 51.92, 49.45}, // ...
   //  ... more lines here: a total of 10
};
int years  = 10; //  ... years
int months = 12; //  ... month
```

```
printCelsius( ?????????????????? );
```

```
void printCelsius( double table[][MONTHS],
                   int    years,
                   int    months)
{
       for (int r = 0; r <  years; r++)
       {    // Print the current row
            for(int c = 0; c <  months; c++)
                cout << (table[r][c] - 32) * 5 / 9 << " ";
            cout << endl;
       }
       cout << endl;
}
```

```
double table [YEARS][MONTHS] =
{
     {48.29, 53.00, 55.97}, //  ... more values here: a total of 12
     {50.35, 51.92, 49.45}, // ...
     //  ... more lines here: a total of 10
};
int years  = 10; //  ... years
int months = 12; //  ... month
```

```
printCelsius( tempTable, years, months );
```

```
void printCelsius( double table[][MONTHS],
                   int    years,
                   int    months)
{
       // ...
}
```

**7.** Write a function that takes a table of quizzes and calculates and stores the class average for each quiz. Call it from main.

```
int quizTable [STU] [QUIZZES] =
{
    {10,  8,  9, 10,  4,  7, 10,  9},
    { 7, 10,  8,  9,  6,  8,  9, 10},
    // ... more lines here: a total of 46
};

int stu     = 46; // ... students in a class
int quizzes =  8; // ... a total of 8 quizzes
```

## calcQuizAvg(?????????????? );

```
void calcQuizAvg ( int    table[][QUIZZES],
                   int    stu,
                   int    quizzes,
                   double quizAvg[])
{
       int sum;

       for (int c = 0; c < quizzes; c++)        // for each quiz
       {
              sum = 0;
              for(int r = 0; r < stu; r++)      // for each student
              {
                  sum += table[r][c];
              }
              quizAvg[c] = (double) sum / stu;
       }
}
```

```
int quizTable [STU] [QUIZZES] =
{
     {10,  8,  9, 10,  4,  7, 10,  9},
     { 7, 10,  8,  9,  6,  8,  9, 10},
     // ... more lines here: a total of 46
};
int stu      = 46; // ... students in a class
int quizzes =  8; // ... a total of 8 quizzes
double quizAvg[QUIZZES];
```

```
calcQuizAvg( quizTable, stu, quizzes, quizAvg );
```

```
void calcQuizAvg ( int    table[][QUIZZES],
                   int    stu,
                   int    quizzes,
                   double quizAvg[])
{
       // …
}
```

**8.** Write a function that takes a table of quizzes and calculates and stores the highest quiz score for each student. Call it from main.

```
int quizTable [STU] [QUIZZES] =
{
    {10,  8,  9, 10,  4,  7, 10,  9},
    { 7, 10,  8,  9,  6,  8,  9, 10},
    // ... more lines here: a total of 46
};

int stu     = 46; // ... students in a class
int quizzes =  8; // ... a total of 8 quizzes
```

## calcHighStu(?????????????? );

```
void calcHighStu ( int    table[][QUIZZES],
                   int    stu,
                   int    quizzes,
                   double highQuizStu[])
{
       int high;

       for (int r = 0; r < stu; r++)              // for each student
       {
              high = table[r][0];
              for( int c = 1; c < quizzes; c++) // for each quiz
              {
                     if ( table[r][c] > high )
                            high = table[r][c];
              }
              highQuizStu[r] = high;
       }
}
```

```
int quizTable [STU] [QUIZZES] =
{
     {10,  8,  9, 10,  4,  7, 10,  9},
     { 7, 10,  8,  9,  6,  8,  9, 10},
     // ... more lines here: a total of 46
};
int stu      = 46; // ... students in a class
int quizzes =  8; // ... a total of 8 quizzes
double highQuizStu[STU];
```

```
calcHighStu( quizTable, stu, quizzes, highQuizStu );
```

```
void calcHighStu ( int    table[][QUIZZES],
                   int    stu,
                   int    quizzes,
                   double highQuizStu[])
{
       // …
}
```

**9.** Read an array knowing the number of rows and columns.

Write a function that reads a maze from a file into a table.
The text file has the number of rows and columns on the first line followed by the maze on the next lines. The maze consists of characters: `'.'` and `'W'` – wall. Here is an example:

```
8 5
W...W
WWW.W
..W.W
W.W.W
W.W..
W.W.W
W...W
WWWWW
```

```
char   mazeTable[ROWS][COLS];
int    rows; // the actual number of rows
int    cols; // the actual number of columns


// a calling statement for readMaze
if (readMaze( mazeTable, rows, cols, "MAZE.TXT" ))
     // success: process mazeTable
else
     // not success: print an appropriate error message

// write the definition for readMaze below;
// it returns true if success, or false otherwise
bool readMaze( char mazeTable[][COLS],
               int &rows,
               int &cols,
               char myMaze[] )
{
```

**10.** Read array knowing the number of columns but not the number of rows (stop at end of file)

Write a function that reads quiz scores from a file into a table. The text file has a 4 digits student ID followed by 10 quiz scores on each line. The number of students is not stored in the file, but you know that for each student there's a corresponding line in the text file (assume the text file has been checked and that it is has valid data: no incomplete lines, no empty lines, etc.)

Here is an example:

```
1111 10  9 10 10  8 10  9 10 10 10
2222  9  8  7  8  8  9  7 10 10 10
. . .
1234  8  9 10  9  8  9 10  9  9  9
```

```
int scoreTable[STU][QUIZ]; //  assume QUIZ is 10
int stuID[STU];
int sNo; // the actual number of students

//  a calling statement for readQuizScores
readQuizScores( scoreTable, stuID, sNo, QUIZ, "SCORES.TXT" );

//  write the definition  for readQuizScores below
void readQuizScores( int  scoreTable[][QUIZ],
                     int  stuID[],
                     int &sNo,
                     int  quizzes,
                     string myScores )
```

## 2D Arrays and Functions

Examples:

- ✓ 1. Passing one element of a 2D Array by value
- ✓ 2. Passing one element of a 2D Array by reference
- ✓ 3. Passing the entire table
- ✓ 4. Passing one row of a 2D Array
- ✓ 5. Passing the index of a column
- ✓ 6. Fahrenheit to Celsius
- ✓ 7. Class average for each quiz
- ✓ 8. Highest quiz for each students
- ✓ 9. A maze
- ✓ 10. Read data from file into a 2D Array