Multi Dimensional Arrays

Chapter 5

2d, 3d, nested loops

2D Array

Column

Row	0	1	2	3
0	table[0][0]	table[0][1]	table[0][2]	table[0][3]
1	table[1][0]	table[1][1]	table[1][2]	table[1][3]
2	table[2][0]	table[2][1]	table[2][2]	table[2][3]
3	table[3][0]	table[3][1]	table[3][2]	table[3][3]

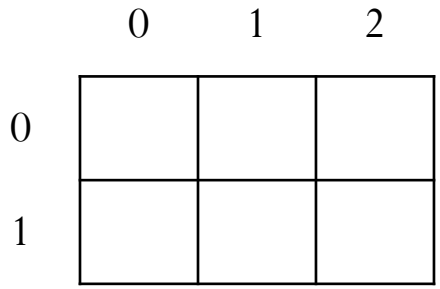
int table[4][4]

Applications

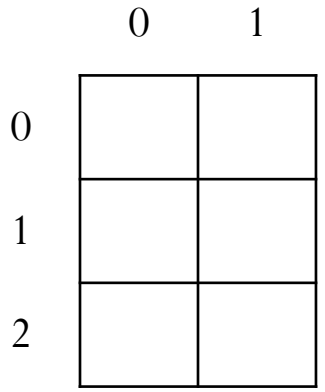
- 2D Arrays can be used for the following applications:
  - Spreadsheet
  - Storing Photos
  - Grade Books
  - Mathematic Matrix Operations

2D Array Declaration

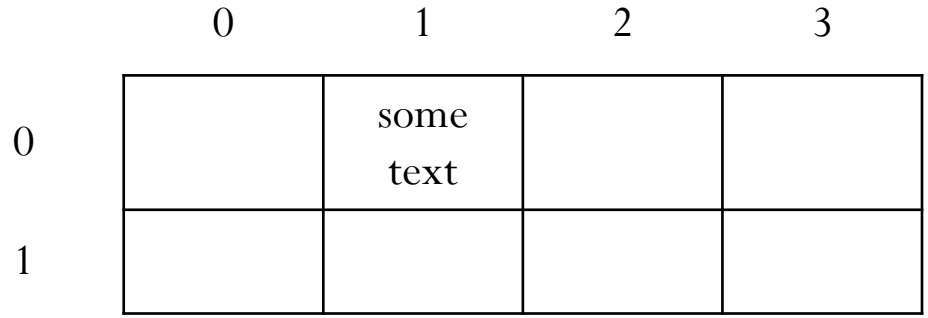
datatype var_name[# rows][# cols];



int x1[2][3]



float x2[3][2]

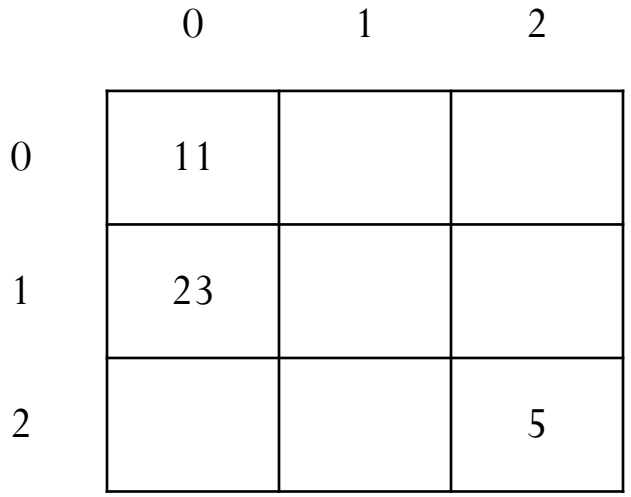


string x3[2][4]

Storing and Fetching

```
int x[3][3];

x[0][0] = 11;
x[1][0] = 23;
x[2][2] = 5;
```




```
cout << x[0][0] << endl;  // Prints: 11
cout << x[1][0] << endl; //  Prints: 23
cout << x[2][2] << endl; //  Prints: 5
cout << x[0] << endl; // Prints a memory address
cout << x << endl; //Prints a memory address
```

Traversing 2D Array

```
int x[3][3] = {{11, 2, 5}, {23, 90, 10}, {12, 26, 5}};
```

Now lets print the array as a table in the console to look like the following:

Table:

```
11      2       5

23      90      10

12      26      5
```

	0	1	2
0	11	2	5
1	23	90	10
2	12	26	5

Traversing 2D Array

```
int x[3][3] = {{11, 2, 5}, {23, 90, 10}, {12, 26, 5}};
```

Lets simplify the problem to one that is familiar, printing a single row.

	0	1	2
0	11	2	5
1	23	90	10
2	12	26	5

Printing row 0:

```
for(j = 0; j < 3; j++)
    cout << setw(3) << x[0][j];
```

Output:

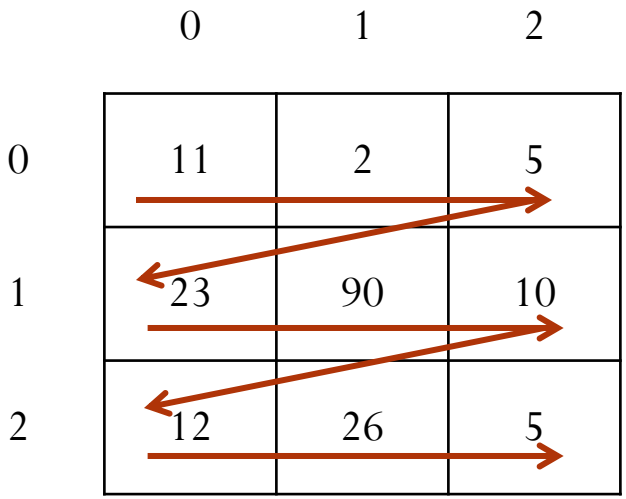
```
11  2  5
```

Traversing 2D Array

```
int x[3][3] = {{11, 2, 5}, {23, 90, 10}, {12, 26, 5}};
```

Generalize to All Rows:

```
for(i = 0; i < 3; i++)
{
   for(j = 0; j < 3; j++)
      cout << setw(3) << x[i][j];
   cout << endl;
}
```



Output:

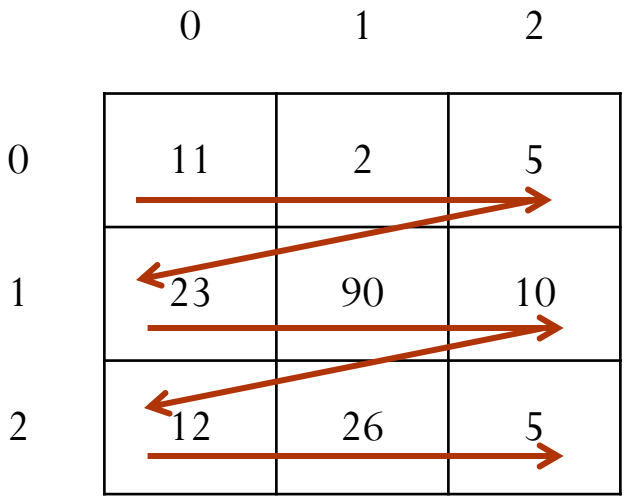
```
 11  2  5
 23 90 10
 12 26  5
```

Example 9.1

```
int x[3][3] = {{11, 2, 5}, {23, 90, 10}, {12, 26, 5}};
```

Generalize to All Rows:

```
for(i = 0; i < 3; i++)
{
   for(j = 0; j < 3; j++)
      cout << setw(3) << x[i][j];
   cout << endl;
}
```



Output:

```
 11  2  5
 23 90 10
 12 26  5
```

Example 9.2 – Adding Matrices

- Create two arrays as shown below:

x

3	54	32
4	98	64
6	23	78

y

7	41	34
3	6	4
55	75	2

- Add the two arrays together as if they were matrices (element by element) and store them into a third matrix

z

3 + 7	54 + 41	32 + 34
4 + 3	98 + 6	64 + 4
6 + 55	23 + 75	78 + 2

z

10	95	66
7	104	68
61	98	8

Example 9.2 – Solution

- Using the same traversal as printing the 2D array in example 9.1 we do the following:

```
// Declaration with Initialization
int x[3][3] = {{3, 54, 32}, {4, 98, 64}, {6, 23, 78}};
int y[3][3] = {{7, 41, 34}, {3, 6, 4}, {55, 75, 2}};
int z[3][3], row, col;

// Traversing 2D Array Element by Element
for(row = 0; row < 3; row++)
{
    for(col = 0; col < 3; col++)
        z[row][col] = x[row][col] + y[row][col];
}
```

Example 9.3 – Tic-Tac-Toe

- Assume you have a tic-tac-toe board represented as a 2D array of characters.
  - A space represents a open position on the board
  - An 'X' as player *x*'s piece
  - An 'O' as player *o*'s piece

- Write routines to do the following:
  - Check if player x has a row filled with x's
  - Check if player x has a column filled with x's
  - Check if player x has a diagonal filled with x's

Testing A Row

Example of game board

	0	1	2
0	X	X	X
1	O		
2		O	

- Below is code testing Row 0:

```
for(col = 0, won = true; col < 3 && won; col++)
        if(board[0][col] != 'X')
           won = false;
```

- We start with the assumption that player X has won, and then iterate through the columns determining if the assumption holds true

Example 9.3 – Tic-Tac-Toe

- Testing Multiple Rows:

```
// Declaration with Initialization
char board[3][3] = {{‘X', ‘X', ‘X'}, {‘O', ' ', ' '}, {‘ ', ‘O',
‘ '}};
int row, col;
bool won;

// Testing Row for Filled X’s
for(row = 0, won = false; row < 3 && !won; row++)
{
    for(col = 0, won = true; col < 3 && won; col++)
        if(board[row][col] != 'X')
           won = false;
}

if(won)
   cout << "X won" << endl;
else
   cout << "X has not won" << endl;
```

Testing A Column

Example of game board

	0	1	2
0	X		O
1	X		
2	X	O	

- Below is code testing Column 0:

```
for(row = 0, won = true; row < 3 && won; row++)
        if(board[row][0] != 'X')
           won = false;
```

- As before we start with an assumption
- Here the row iterates while the column remains fixed

Example 9.3 – Tic-Tac-Toe

- Testing Columns:

```
// Declaration with Initialization
char board[3][3] = {{‘X', ' ', ‘O'}, {'X', ' ', ' '}, {'X', ‘O',
‘ '}};
int row, col;
bool won;

// Testing Row for Filled X’s
for(col = 0, won = false; col < 3 && !won; col++)
{
    for(row = 0, won = true; row < 3 && won; row++)
        if(board[row][col] != 'X')
           won = false;
}

if(won)
   cout << "X won" << endl;
else
   cout << "X has not won" << endl;
```

Testing A Diagonal UL to LR

Example of game board

	0	1	2
0	X	O	
1	O	X	
2		O	X

- Test Locations (row, col)

(0,0), (1,1), (2,2)

- Notice the index of the columns and rows stay the same
  - Thus we can represent both the row and column as a single variable
  - Only one loop is required in this case

```
// Testing Diagonal NW to SE for X's
for(i = 0, won = true; i < 3 && won; i++)
{
    if(board[i][i] != 'X')
        won = false;
}
```

Example 9.3 – Tic-Tac-Toe

- Testing Diagonal from Upper Left to Lower Right:

```
// Declaration with Initialization
char board[3][3] = {{'X', 'O', ' '}, {'O', 'X', ' '}, {' ', 'O',
'X'}};
int i;
bool won;

// Testing Diagonal NW to SE for X's
for(i = 0, won = true; i < 3 && won; i++)
{
    if(board[i][i] != 'X')
        won = false;
}

if(won)
    cout << "X won" << endl;
else
    cout << "X has not won" << endl;
```

Testing A Diagonal LL to UR

Example of game board

	0	1	2
0		O	X
1	O	X	
2	X	O	

- Test Locations (row, col)

(2,0), (1,1), (0,2)

- Notice the index of the column increases while the row decreases
  - We can represent these as two variables (if ur clever as one variable)
  - For each row there is only one column to visit therefore a nested loop is not required

```
// Testing Diagnol SW to NE for X's
for(row = 2, col = 0, won = true; col < 3 && won; row--, col++)
{
   if(board[row][col] != 'X')
       won = false;
}
```

Example 9.3 – Tic-Tac-Toe

- Testing Diagonal from Lower Left to Upper Right:

```
// Declaration with Initialization
char board[3][3] = {{‘ ', ‘O', 'X'}, {‘O', 'X', ' '}, {'X', ‘O',
‘ '}};
int row, col;
bool won;

// Testing Diagnol SW to NE for X's
for(row = 2, col = 0, won = true; col < 3 && won; row--, col++)
{
   if(board[row][col] != 'X')
       won = false;
}

if(won)
    cout << "X won" << endl;
else
    cout << "X has not won" << endl;
```

Multi-Dimensional Arrays

- 3D 2x2x2

```
int x[2][2][2];
```

- 4D 2x2x2x2

```
int x[2][2][2][2];
```

- nD 2x2x … x2

```
int x[2][2]…[2];
```

Beware: Size of the array increases exponentially with respect to the number of dimensions as such so does the memory

Printing the contents of a 3D Array

```
// Declaration with Initialization
int cube[3][3][3];
int depth, row, col;

for(depth = 0; depth < 3; depth++)
{
    for(row = 0; row < 3; row++)
    {
        for(col = 0; col < 3; col++)
        {
            cout << cube[depth][row][col] << endl;
        }
    }
}
```

Printing n-dimensional array

- Requires n nested for loops

```
for(x1)
  for(x2)
    for(x3)
      …
        for(n)
```

- Using pointers you can flatten the array to 1D and use a single loop but that is a topic for another time.