Chapter 13
Vector of Vectors (2D Arrays)

3rd Edition

Computing Fundamentals with C++

Rick Mercer
Franklin, Beedle & Associates

Goals

- Process data stored as a vector of vectors (rows and columns)
- Use nested for loops
- Show a couple matrix operations

Vector of Vectors

- Data that conveniently presents itself in a tabular format is represented well with a `vector` of vectors
- General form

Space required

```
vector <vector<type> > identifier(rows,
          vector<type> (cols, initialValue optional ));
```

- Example declaration (the space is needed > >)

```
vector <vector<int> > nums(5,
          vector<int> (3, -1));
// All 15 values (5 rows, 3 columns) are -1
```

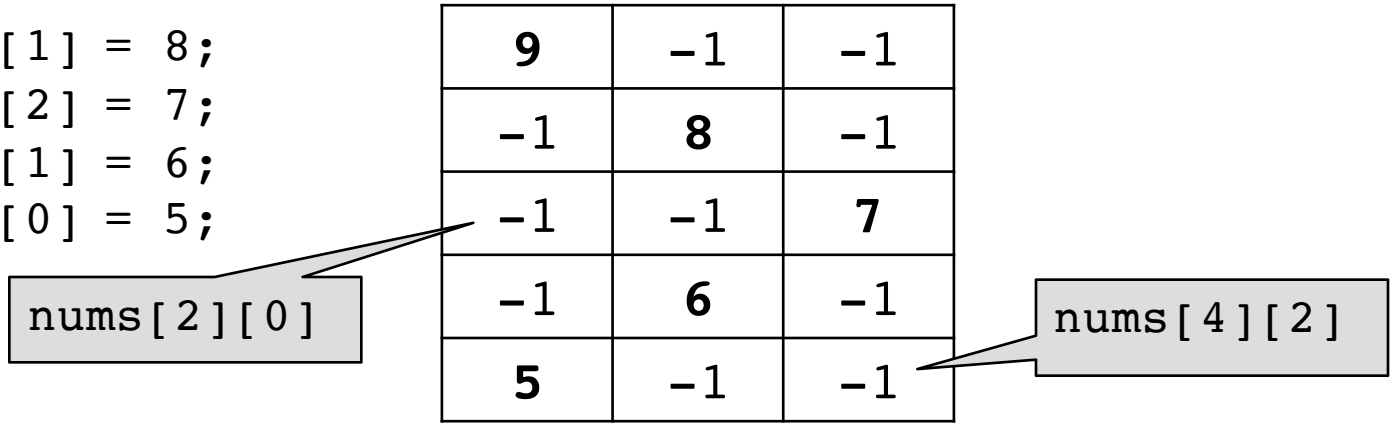
- Set the value of the first row and column to 9

```
nums[0][0] = 9;
```

Rows and columns

- Change a few more elements

```
nums[1][1] = 8;
nums[2][2] = 7;
nums[3][1] = 6;
nums[4][0] = 5;
```



- Nested for loops are often used with a vector of vectors

```
for (int row = 0; row < nums.size(); row++) {
  for (int col = 0; col < nums[0].size(); col++) {
    cout.width(4);
    cout << nums[row][col];
  }
  cout << endl;
}
```

Output:

```
   9  -1  -1
  -1   8  -1
  -1  -1   7
  -1   6  -1
   5  -1  -1
```

Matrix Operations

- A matrix is a rectangular vector of numbers, symbols, or expressions, arranged in rows and columns
- The data structure best suited to represent a matrix in C++? A vector of vectors
- This presentation shows of a couple of the operations on this data structure

Build a 6x6 Matrix

```
vector<vector<int> > nums(6, vector<int>(6));
 // all 36 values (6 rows, 6 columns) are garbage

 // Initialize to arbitrary values: 1..36
 int count = 1;
 for (int row = 0; row < nums.size(); row++) {
   for (int col = 0; col < nums[0].size(); col++) {
     nums[row][col] = count;
     count++;
   }
 }
```

Matrix:

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

Scalar Multiplication

- Scalar multiplication takes a number (a "scalar") and multiplies it on every element in the matrix

```
int scalar = 2; // Multiply all elements by 2
for (int row = 0; row < nums.size(); row++) {
  for (int col = 0; col < nums[0].size(); col++) {
    nums[row][col] *= scalar;
  }
}
```

Matrix:

2	4	6	8	10	12
14	16	18	20	22	24
26	28	30	32	34	36
38	40	42	44	46	48
50	52	54	56	58	60
62	64	66	68	70	72

Transpose the Matrix

- The transpose of a Matrix makes all row the columns and all columns the rows

```
// Create a transposed version
vector<vector<int> > transpose(6, vector<int>(6));
for (int row = 0; row < nums.size(); row++) {
  for (int col = 0; col < nums[0].size(); col++) {
    transpose[col][row] = nums[row][col];
  }
}

// Copy the transposed matrix back into nums
for (int row = 0; row < nums.size(); row++) {
  for (int col = 0; col < nums[0].size(); col++) {
    nums[row][col] = transpose[row][col];
  }
}
```

Before and After

- The first row (before) is now the first column (after)
- The last row (before) is now the last column (after)
- The 2nd row (before) is now the 2nd column (after)
- . . . and so on

before

2	4	6	8	10	12
14	16	18	20	22	24
26	28	30	32	34	36
38	40	42	44	46	48
50	52	54	56	58	60
62	64	66	68	70	72

after

2	14	26	38	50	62
4	16	28	40	52	64
6	18	30	42	54	66
8	20	32	44	56	68
10	22	34	46	58	70
12	24	36	48	60	72

Primitive Arrays

- There are similarities between vector and primitive C++ arrays
- There are also similarities between a vector of vectors and primitive C++ arrays with two subscripts

```
vector<vector<int> > table(4,
        vector<int>(6)); //   Ugh

string table[4][6]; // Simpler declaration
```

- Elements are accessed the same way

```
table[0][0]= "Upper left";
table[3][5]= "Lower right";
```

Array Initializers

- Primitive arrays are easier to declare
  - feel free to use them instead of vectors
  - A lot of code uses the primitive C++ arrays
- Arrays also have initializers, values between { } separated by commas

```
double oneSubscript[] = {1.5, 0.9, 0.03, 4.2};

int twoSubscripts[2][3] = { {1, 2, 3}, {4, 5, 6} };
```

Arrays with more than 2 Subscripts?

- Yes, they are possible and sometimes useful

```
double threeD[2][35][10];

threeD[1][34][9] = 87;
```