Multi Dimensional Arrays

Storage Order: It is the way in which a multi-dimensional array is stored in system memory. There are two major ways to do that

1) Row Major (most used in programming languages)
2) Column Major (only FOTRAN)

System memory is a linear address space. So multi-dimensional array addresses must be converted to linear addresses before accessing in memory

Consider a 3D array a[m][n][p]. An element of this array can be addressed by a[i][j][k].

In a row-major ordered system next element can be accessed at a[i][j][k+1] while in a column-major ordered system next element can be accessed at a[i+1][j][k]

Declaration: double matrix[][] = new double[n][m];

Declaration in a loop:

```
for(…)
{
        double matrix[][] = new double[n][m];
        .
        .
        .
}
```

In every iteration we loose the array object and the reference variable matrix. Every time a new reference variable is created.

Note: Declaring the same variable more than once is an error normally. But the above loop it is not an error. The number of times the loop runs is got at runtime

Another way of declaring in a loop:

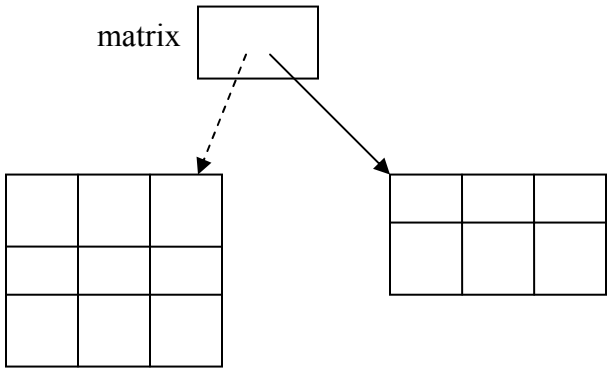
```
double matrix[][];

for(…)
{
        matrix = new double[n][m];
        .
        .
        n = n+1;
}
```

Here the object assigned to matrix reference variable is changing every time. Each time n and m can be different.

Note: Once the array declaration is done the value of n can be changed independent of the previous declaration. All the statements are individually evaluated.

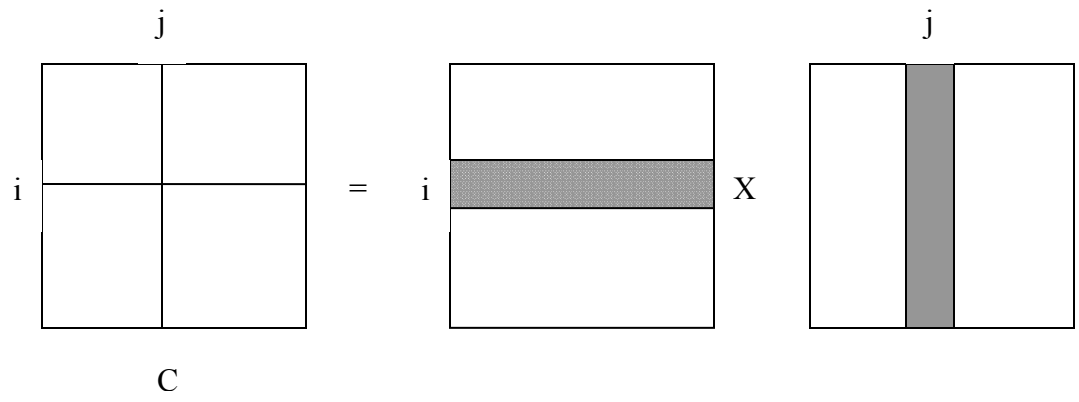
Note: We cannot have two array variables of different size.



Only one arrow is active at a time. i.e., matrix variable stores reference to one array at a time.

Matrix Multiplication:

Multiplication of matrices results in a matrix. We have to calculate n*m values. Where each value is dot product of vectors from given matrices



$$C_{ij} = \sum_{k=1}^{n} A_{ik} * B_{kj}$$

```
for ( i = 0; i < n; i++ )
        for ( j = 0; j < n; j++ )
        {
                c[i][j] = 0.0;
                for( k = 0; k < n; k++ )
                        c[i][j] += a[i][k] * b[k][j];
        }
```

More Operators

Bitwise AND (&) : performs bit-by-bit AND

Example: 5 & 3 ≡ 101 & 011 = 001 ≡ 1

Usage: c = a & b;

Similarly, Bitwise OR (|) and Bitwise XOR(^). All these operators don't have side effects i.e., a and b are not changed while evaluating the expression

Folding Operators: These operators have an operation and assignment included in them

Examples: +=, -=, *=, /=, %=

Read LHS value; Evaluate RHS value; do operation; Store in LHS

These operators have lowest precedence of all the operators.

Rule of Thumb: Don't use two disjoint side effects in a statement. Never have two conflicting side effects in an expression.

"c[i] = a[i] + b[i++]" is a badly formed expression.

a=a!=b is a bad expression, to deal with such expression use parenthesis. (a=(a!=b))

Sudoku Application

Data Structures:

Cell[][] sudoku = new Cell[9][9];

Cell should be able to store the list of digits (among 9) that can be placed in that cell. This can be achieved by a boolean array of size 9.

```
class Cell
{
        private boolean[] contents;

        Cell()  // default constructor
        {
                contents = new boolean[9];
                for( int i = 0; i < 9; i++ )
                        contents[i] = true;  // empty cell should be able to
store any number
        } // end constructor

        Cell( Cell c )
        {
                contents = new boolean[9];
                for( int i = 0; i < 9; i++ )
                        contents[i] = c.contents[i];  // copy values
        } // end constructor

        Cell( int i)
        {
                if( (i<1) || (i>Sudoku.N) )
                        <code same as in default constructor>
```

```
                else
                {
                        contents = new boolean[9];
                        contents[i-1] = true;
                }
        } //end constructor
} // end class
```