Multidimensional Array

Declaration

- Higher dimensional arrays are also supported.
- Declaration of such an array could: `int a[5][10];`
- Multidimensional arrays are considered as array of arrays.
- Such array are programming abstraction, storage allocation remains same.

Multidimensional Array

Declarations & Initializations

```
#include <stdio.h>
int main() {
   int a[5][10];
   int i, j;

   for (i = 0; i < 5; i++)
      for (j = 0; j < 10; j++)
          a[i][j] = (i+1) * (j+1);
}
```

```
int main() {
   int a[50];
   int i, j;

   for (i = 0; i < 5; i++)
      for (j = 0; j < 10; j++)
          a[i*10+j] = (i+1) * (j+1);
}
```

Multidimensional Array

Matrix Operation

- Now examine operations on multidimensional arrays.
- **Problem**: Generate and print a 2-D matrix with elements having integral values.
- Random number generator is used to generate each element.
- Elements are stored in matrix.
- Each row of matrix in a separate line.

Multidimensional Array

Generate Matrix

```
void GenerateMatrix(double a[][MAX], int k) {
   int i, j;

   srand((unsigned int) time(NULL)); // Seed the generator

   for (i = 0; i < MAX; i++)
       for (j = 0; j < MAX; j++)
           a[i][j] = (double)(rand() % k);  // Generate values
}
```

Multidimensional Array

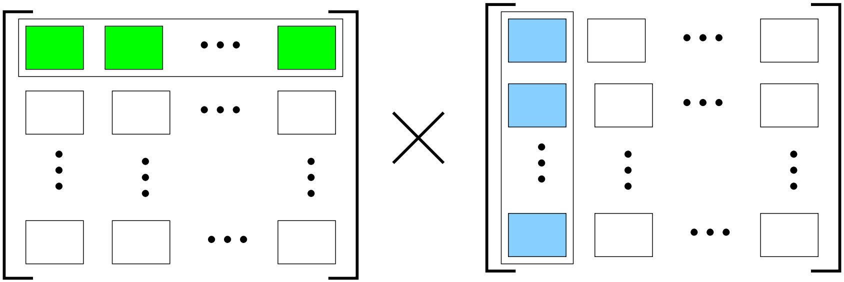
Printing Matrix

```
void PrintMatrix(double a[][MAX]) {
    int i, j;

    for (i = 0; i < MAX; i++) {
        for (j = 0; j < MAX; j++)
            printf("%g\t", a[i][j]); // Print a row in one line
        printf("\n"); // New line
    }
}
```

Multidimensional Array

Multiplying Matrices



Multidimensional Array

Multiplying Matrices

```
void MatProd(double a[][MAX], double b[][MAX], double c[][MAX]) {
    int i, j, k;

    for(i = 0; i < MAX; i++)
        for(j = 0; j < MAX; j++) {
            c[i][j] = 0.0; // Initialize elements of product matrix
            for(k = 0; k < MAX; k++)
                c[i][j] += a[i][k] * b[k][j]; // Inner product
        }
}
```

Multidimensional Array

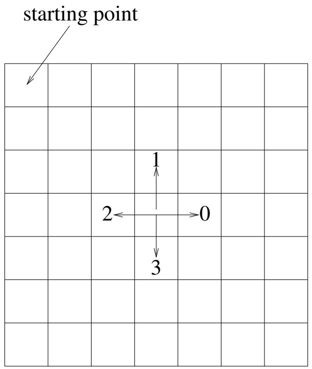
Multiplying Matrices

```c
#include <stdio.h>
#include <time.h>
int main() {
   double a[MAX][MAX], b[MAX][MAX], c[MAX][MAX];

   GenerateMatrix(a, 10);
   sleep(1); // Make sure generated elements are different
   GenerateMatrix(b, 10);
   PrintMatrix(a);
   PrintMatrix(b);
   MatProd(a, b, c);
   PrintMatrix(c);
}
```

Multidimensional Array

Traversing a Checker Board



- Possible moves: left, up, right and down.
- Initial move: picked randomly from 4 possibilities.
- If no vacant slot is available, then stop.

Multidimensional Array

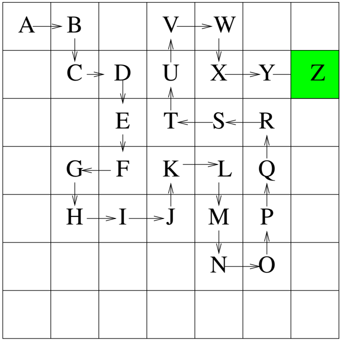
Traversing a Checker Board

- Start at top left corner and place character 'A'
- Determine the next slot to move by random number (0..3).
- Place character 'B' in the slot, this becomes the current slot.
- Again determine next slot by random number.
- If the next slot is occupied find cyclically next slot continue until a vacant slot is found.
- If no neighboring slot is vacant or 26 slots are filled then stop.

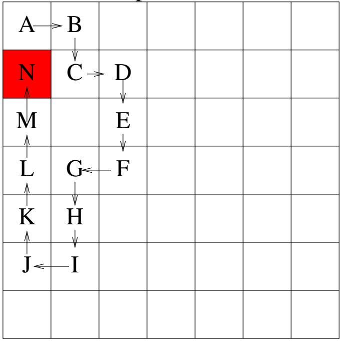
Multidimensional Array

Traversing a Checker Board

A complete trail



A partial trail



Multidimensional Array

Major Steps of Traversal

1. Generate a random number between 0..3
2. Identify the neighboring vacant slot from the current slot.
3. Place next character in the vacant slot determined in step 3.
4. Repeat from step 1 until character 'Z' has been placed.

Multidimensional Array

Neighboring Slot

- Traversal direction: $\rightarrow$, $\uparrow$, $\leftarrow$, $\downarrow$
- Suppose the current slot is (i, j), to find next vacant slot
  - Generate randomly a number in{ 0, 1, 2, 3}.
  - The sequence of search for vacant slot:

$$\text{Next slot} = \begin{cases} \{(i, j+1), (i-1, j), (i, j-1), (i+1, j)\}, \text{ if } k = 0 \\ \{(i-1, j), (i, j-1), (i+1, j), (i, j-1)\}, \text{ if } k = 1 \\ \{(i, j-1), (i+1, j), (i, j-1), (i-1, j)\}, \text{ if } k = 2 \\ \{(i+1, j), (i, j-1), (i-1, j, (i, j-1))\}, \text{ if } k = 3 \end{cases}$$

If all 4 slots are occupied or non-existent then no vacant slot

Multidimensional Array

Finding Free Slot

```
int freeSlot(char a[][8], int n, int i, int j) {
    int k, l = 0;

    k = rand() % 4 ;
    while (l < 4 ) { // Exit loop: checked 4 cyclic slots.

        /* 1. For k = 0, check if j+1 > N-1 or a[i, j+1]       *
         *    is nonempty, then slot (i, j+1) is not useable   *
         * 2. For k = 1, check if i-1 < 0 or a[i-1, j]         *
         *    is nonempty, then slot (i-1, j) is not useable   *
         * 3. Similarly check other two border cases.          */
    }
    return -1; // No free slot available
}
```

Multidimensional Array

Finding Free Slot

```
if (k == 0) {
    if (j + 1 > N-1  || a[i][j+1] != '.') {
        k = (k + 1) % 4; // Should check cyclically next slot
        l++;             // Count the slots checked
    } else // Slot to right is empty
          return k;
} else if (k == 1) {
    if (i - 1 < 0 || a[i-1][j] != '.') {
        k = (k + 1) % 4;
        l++;
    } else  // Slot up is empty
          return k;
}
// Remaining part in the next slide
```

Multidimensional Array

Finding Free Slot

```
else if (k == 2) {
    if (j - 1 < 0 || a[i][j-1] != '.') {
        k = (k + 1) % 4;
        l++;
    } else   // Slot to left is empty
        return k;
} else if (k == 3)  {
    if (i + 1 > N-1 || a[i+1][j] != '.') {
        k = (k + 1) % 4;
        l++;
    } else   // Slot down is empty
        return k;
}
```

Multidimensional Array

Placing Character in Free Slot

```
void checkerBoard(char a[][8], int n) {
    int i = 0, j = 0, l, k;
    char c = 'A';
    a[0][0] = c;

    /* Determine the free slot and place the
       next character in that slot.          */

    return; // All characters have been placed
}
```

Multidimensional Array

Placing Character in Free Slot

```
while (c < 'Z') {
    k = freeSlot(a, n, i, j); // Get next free slot
    if (k == -1) // Slots in all directions occupied
        return;  // All characters not placed.
    c++;
    if (k == 0) { // Slot to right is free
        a[i][j+1] = c; j++;
    } else if (k == 1) { // Slot up is free
        a[i-1][j] = c; i--;
    } else if (k == 2) { // Slot to left is free
        a[i][j-1] = c; j--;
    } else { // Slot down is free
        a[i+1][j] = c; i++;
    }
}
```