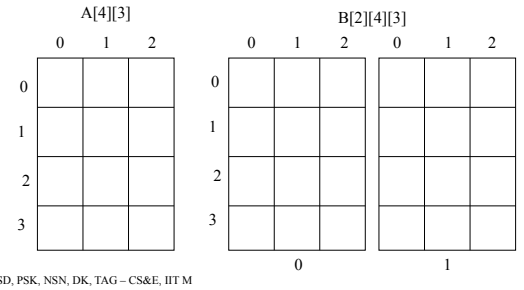
CS1100
Introduction to Programming

Matrix Operations

Madhu Mutyam
Department of Computer Science and Engineering
Indian Institute of Technology Madras

Multi-Dimensional Arrays

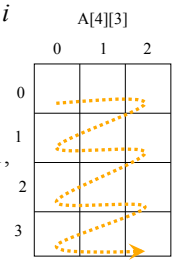
- Arrays with two or more dimensions can be defined



Two Dimensional Arrays

- Declaration: int A[4][3] : 4 rows and 3 columns, 4×3 array
- Elements: A[*i*][*j*] - element in row *i* and column *j* of array A
- Rows/columns numbered from 0
- Storage: **row-major ordering**
  - elements of row 0, elements of row 1, etc
- Initialization:

  int B[2][3]={{4,5,6},{0,3,5}};



Matrix Operations

- An *m*-by-*n* matrix *M*: *m* rows and *n* columns
- Rows: 1, 2, … , *m* and Columns: 1, 2, … , *n*
- *M*(*i*, *j*): element in i^{th} row, j^{th} col., $1 \le i \le m$, $1 \le j \le n$
- Array indexes in C language start with 0
- Use $(m+1)\times(n+1)$ array and ignore cells $(0,i)$, $(j,0)$
- Programs can use natural convention – easier to understand
- Functions: matRead(a,int,int), matWrite(a,int,int), matAdd(a,b,c,int,int), matMult(a,b,c,int,int,int);

Using Matrix Operations

```
main( ){
   int a[11][11], b[11][11], c[11][11]; /*max size: 10 by 10 */
   int aRows, aCols, bRows, bCols, cRows, cCols;
   scanf("%d%d", &aRows, &aCols);
   matRead(a, aRows, aCols);
   scanf("%d%d", &bRows, &bCols);
   matRead(b, bRows, bCols);
   matMult(a, b, c, aRows, aCols, bCols);
   cRows = aRows; cCols = bCols;
   matWrite(c, cRows, cCols);
}
```

Address of the (0,0) element of the array

Remember bRows=aCols

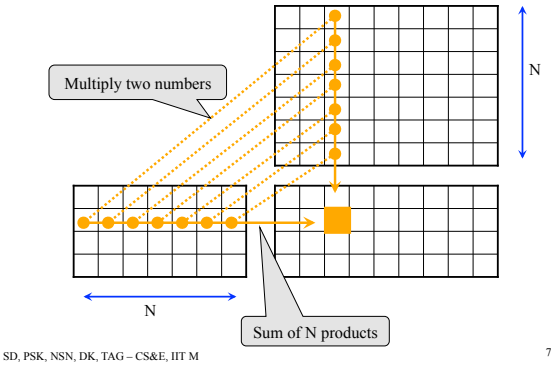
Reading and Writing a Matrix

```
void matRead(int mat[ ][11], int rows, int cols){
   for (int i = 1; i <= rows; i++)
     for (int j = 1; j <= cols; j++)
        scanf("%d", &mat[i][j]);
}

void matWrite(int mat[ ][11], int rows, int cols){
   for (int i = 1; i <= rows; i++){
     for (int j = 1; j <= cols;  j++)    /* print a row */
        printf ("%d ", mat[i][j]);    /* notice missing \n */
     printf ("\n");              /* print a newline at the end a row */
   }
}
```

For the compiler to figure out the address of mat[i][j], the first dimension value is not necessary. (Why?)

Matrix Multiplication



Matrix Multiplication

```
void matMult(int mat1[ ][11], int mat2[ ][11],
             int mat3[ ][11], int m, int n, int p){
for (int i =1; i <= m; i++)
    for (int j = 1; j <= p; j++)
        for (int k = 1; k <= n; k++)
            mat3[i][j] += mat1[i][k]*mat2[k][j];
}
```

Remember it was initialized to zero in the main program. It could have been done in this function as well – probably a better idea.

scanf and getchar

- getchar( ) reads and returns one character
- scanf – formatted input, stores in variable
  - scanf *returns an integer* = number of inputs it managed to convert successfully

```
printf ("Input 2 numbers: ");
if (scanf ("%d%d", &i, &j) == 2)
        printf ("You entered %d and %d\n", i, j);
else printf ("You failed to enter two numbers\n");
```

from <http://cprogramming.com>

Input Buffer

- Your input line is first stored in a buffer
- If you are reading a number with ***scanf*** (%d) and enter 1235ZZZ, ***scanf*** will read 1235 into the variable and *leave ZZZ in the buffer*
- The next read statement will get ZZZ and may ignore the actual input!
- One may need to write a statement to clear the buffer…

```
while (getchar( ) != '\n');
```

This reads and ignores input till the end of line

Program to Insist on One Number Only

```
#include <stdio.h>
int main(void){
  int temp;
  printf ("Input your number: ");
  while (scanf("%d", &temp) != 1)
    {
      while (getchar( ) != '\n');
      printf ("Try again: ");
    }
  printf ("You entered %d\n", temp);
  return(0);
}
```

exit if one number

clear buffer before reading again

Experiments with Numbers

- The Collatz problem asks if iterating

$$\alpha_n = \begin{cases} \frac{1}{2}\alpha_{n-1} & \text{for } \alpha_{n-1} \text{ even} \\ 3\alpha_{n-1} + 1 & \text{for } \alpha_{n-1} \text{ odd} \end{cases}$$

always returns to 1 for positive α. The members of the sequence produced by the Collatz problem are sometimes known as *hailstone numbers*.

From Wolfram Mathworld
http://mathworld.wolfram.com/CollatzProblem.html

Hailstone Numbers

- A Hailstone Sequence is generated by a simple algorithm:

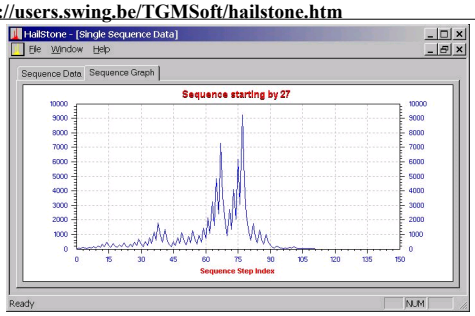
Start with an integer *N*. If *N* is even, the next number in the sequence is *N/2*. If *N* is odd, the next number in the sequence is (*3***N*)+1

- 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, ... *repeats*
- 12, 6, 3, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1 ….
- 909, 2726, 1364, 682, 341, 1024, 512, 256, 128, 64, 32, 16, 8, 4, 2, 1, 4, 2, 1…

2^{10}

Mathematical Recreations

http://users.swing.be/TGMSoft/hailstone.htm



Exercise : Write a program to accept an input and count the number of iterations needed to get to 1, and the highest number reached. Generate a table of results…