

# CSE 142 Computer Programming I

## Multidimensional Arrays

© 2000 UW CSE

R-1

## Overview

### Review

1-D arrays

### Concepts this lecture:

2-D arrays

2-D arrays as parameters

Layout of 2-D arrays in memory

### Reading

Textbook sec. 8.7

R-2

## Arrays as Data Structures

*Review: An array is an ordered collection of values of identical type*

**Name** the collection; **number** the elements

Arrays are the natural choice for organizing a large number of values, all of identical type

R-3

## Beyond Simple Arrays

Sometimes the collection of values has some additional regular pattern or structure

One common such structure is the matrix or table

In C, we can express this as a two-dimensional array

Higher-dimensional arrays (3-D, 4-D, ...) are possible, but we won't use them in this course

R-4

## 2-Dimensional Arrays

*An ordered collection of values of identical type*

**Name** the collection; **number** the elements

Like 1-D arrays, but a different numbering scheme

Example: scores for 7 students on 4 homeworks

score	hw	0	1	2	3
student 0		22	15	25	25
student 1		12	12	25	20
student 2		5	17	25	24
student 3		15	19	25	13
student 4		2	0	25	25
student 5		25	22	24	21
student 6		8	4	25	12

C expressions:

`score[0][0]` is 22

`score[6][3]` is 12

`2*score[3][0]` is 30

R-5

## Declaring a 2-D Array

```
#define MAX_STUDENTS 80
```

```
#define MAX_HWS 6
```

...

```
int score [MAX_STUDENTS] [MAX_HWS] ;
```

R-6

## 2-D Arrays: Terminology

`type name[#rows][#columns]`

```
int score[80][6];
```

*score is a two-dimensional array of int of size 80 by 6*

*score[0][0], score[0][1], ..., score[79][5] are the elements of the array*

R-7

## An Alternate View

```
int score[80][6];
```

We could also view each row as an element:

“score is an array of size 80”

With this view, each element (row) is a 1-D array, of type “array of size 6 of int”

R-8

## Bookkeeping

As with 1-D arrays, often we only use part of the space available in a 2-D array

*Declared size of the array specifies its maximum capacity.*

*The current size (# of rows and columns currently in use) needs to be kept track of in separate variables*

R-9

## Reading in Data

Problem: Read in data for student assignments

Input data format: The number of students, then the number of assignments, followed by the data per student

A nested loop is the right program structure for reading in the data details

```
int score [MAX_STUDENTS] [MAX_HWS] ;  
int nstudents, nhws, i, j ;
```

R-10

## Reading a 2-D Array: Code

```
/* Read the number of students and assignments,  
then loop to read detailed data */
```

```
scanf ("%d %d", &nstudents, &nhws) ;  
if (nstudents <= MAX_STUDENTS &&  
    nhws <= MAX_HWS) {
```

```
    for ( i = 0 ; i < nstudents ; i = i + 1 )
```

```
        for ( j = 0 ; j < nhws ; j = j + 1 )
```

```
            scanf ("%d", &score [i] [j]) ;
```

```
    }
```

R-11

*Part of the array is unused; which part?*

## Array Input Trace

```
Input: 7 4 0 1 2 3 4 5 6 7 8 9 ...
```

score	j=	0	1	2	3	4	5	...
i=0		0	1	2	3	?	?	...
i=1		4	5	6	7	?	?	...
i=2		8	9	...				
...		...						
i=6		...						
i=7		?	?	?	?	?	?	...

R-12

## Printing a 2-D Array

```
if (nstudents <= MAX_STUDENTS &&
    nhws <= MAX_HWS) {
    for (i = 0 ; i < nstudents ; i = i + 1) {
        for (j = 0 ; j < nhws ; j = j + 1)
            printf("%d", score [ i ] [ j ] );
        printf("\n");
    }
}
```

R-13

## 2-D Arrays as Parameters

Same as 1-D arrays (almost):

- Individual array elements can be either value or pointer parameters
- Entire arrays are always passed as pointer parameters - never copied
- Don't use & and \* with entire array parameters

Difference:

- No empty brackets [ ] in formal parameters
- Actually, [ ] allowed sometimes; we won't use in this course

R-14

## 2-D Array As Parameter

A function to read into **array a** the grade information for the given number of students and assignments

```
void read_2D (int a [MAX_STUDENTS] [MAX_HWS],
             int nstudents, int nhws)
{...
```

R-15

## 2-D Array As Parameter

*/\* Read into **array a** the grade information for \*/*  
*/\* the given number of students and assignments \*/*

```
void read_2D (int a [MAX_STUDENTS] [MAX_HWS],
             int nstudents, int nhws)
{
    int i, j;
    for (i = 0 ; i < nstudents ; i = i + 1)
        for (j = 0 ; j < nhws ; j = j + 1)
            scanf("%d", &a [ i ] [ j ] );
}
```

R-16

## Array Function Arguments

```
int main(void)
{
    int score [MAX_STUDENTS] [MAX_HWS] ;
    int nstudents, nhws ;

    scanf ("%d %d", &nstudents, &nhws) ;
    if ( nstudents <= MAX_STUDENTS &&
        nhws <= MAX_HWS)
        read_2D (score, nstudents, nhws) ;
    ...
}
```

no &

R-17

## Example - Digital Image

A *digital image* is a rectangular grid of *pixels*

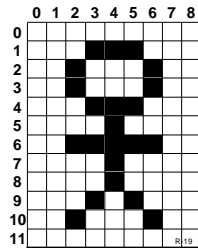
Pixel representation: integer value giving  
brightness from 0 (off) to 255 (full on)  
Black & White: one int per pixel  
Color: 3 ints per pixel - one each for **red**,  
**green**, and **blue**

An image is normally stored as a 2D array

R-18

## Problem - Shift Image

Write a function that shifts a B&W image right one pixel  
right one pixel  
Strategy: shift columns one at a time  
To shift a column, shift its pixels 1 row at a time



## A couple of definitions

```
/* Number of rows and columns in image */
#define NROWS 768
#define NCOLS 1024

/* Representation of a white pixel */
#define WHITE 255
```

R-20

## Code

```
/* Shift image right one column */
void shift_right(int image[NROWS][NCOLS]) {
    int row, col;

    /* shift all columns */
    for (col = ... ) {
        /* shift column col one space to the right */
        for (row = 0; row < NROWS; row++)
            image[row][col+1] = image[row][col];
    }

    /* set leftmost column to white */
    for (row = 0; row < NROWS; row++)
        image[row][0] = WHITE;
}
}
```

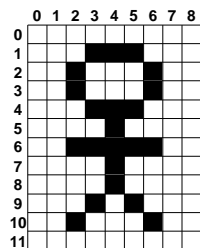
R-21

## Column Sequence

Question: Does it matter if we shift from left to right vs right to left?  
Question: What are the correct loop bounds for col?  
0 to NCOLS-1? 0 to NCOLS-2? 1 to NCOLS-1?  
Something else?

```
/* shift all columns */
for (col = ... ) {
    /* shift column col one space to the right */
    for (row = 0; row < NROWS; row++)
        image[row][col+1] = image[row][col];
}
}
```

R-22



R-23

## Order

Answer: Yes it does. If we shift from left to right, we wind up making copies of column 0 in every column of the image. Correct code looks like this.

```
/* shift all columns */
for (col = NCOLS - 2; col >= 0; col-- ) {
    /* shift column col one space to the right */
    for (row = 0; row < NROWS; row++)
        image[row][col+1] = image[row][col];
}
}
```

R-24

## Image Scrolling

Finally, we can use our function to scroll an image completely off the screen

```
int main(void) {
    int image[NROWS][NCOLS]; /* the image */
    int k;

    /* assume image is initialized elsewhere */
    ...
    /* scroll image completely off the screen */
    for (k = 0; k < NCOLS; k++)
        shift_right(image);

    return 0;
}
```

Note: no & (it's an array)

R-26

## Representation of Arrays

A computer's memory is a one dimensional array of cells

How is a 2-D array stored?

Answer: In C, the array rows are stored sequentially: row 0, 1, 2, ...

## Representation of Arrays

score	hw	0	1	2	3	4	5
student 0		13	15	25	25	?	?
student 1		12	12	25	20	?	?
student 2		5	17	25	24	?	?
student 3		15	19	25	13	?	?
student 4		2	0	25	25	?	?
student 5		25	22	24	21	?	?
student 6		8	4	25	12	?	?
student 7		?	?	?	?	?	?

13	15	25	25	?	?	12	12	25	20	?	?	5	17	25	24	?	?	15	...
← student 0				← student 1				← student 2											

R-27

R-28

## Summary

2-D arrays model matrices or tables of data

Notation and use is an extension of 1-D arrays

Nested loops are often the natural processing technique