INFORMATION & COMPUTER SCIENCE DEPARTMENT

ICS 103 – LAB #10 2-D Arrays

# Objectives:

To gain experience with:

* Declaring and accessing 2D-arrays
* Manipulating 2D-arrays.
* Using 2D-arrays in function calls

# 1. Brief Review of 2-D Arrays

**Why 2D-arrays?**

A 2D-array is used to store information that would otherwise require several 1D-arrays to store. For example, assume you have a table of student grades in 5 quizzes as follows:

**This table can be stored as:**

1. five 1D arrays, each represents a column,
2. six 1D arrays, each represents a row,
3. or one 2D array, in which the position of each element is determined by its row and column

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** |  | **quiz1** | **quiz2** | **quiz3** | **quiz4** | **quiz5** |
| 900000 |  | 50.5 | 40.0 | 60.0 | 0.0 | 55.0 |
| 920000 |  | 70.0 | 60.0 | 75.0 | 90.0 | 66.5 |
| 930520 |  | 65.0 | 70.0 | 65.0 | 80.0 | 78.0 |
| 940000 |  | 80.0 | 90.0 | 95.0 | 85.0 | 100.0 |
| 953478 |  | 40.0 | 30.0 | 50.0 | 55.0 | 45.0 |
| 972893 |  | 60.0 | 50.0 | 39.0 | 70.0 | 55.9 |

The third solution (having a single entity representing 2D array) is more general and convenient for dealing with data that are naturally organized in tables.

In C, a table may be *conceptually* implemented as a 2D array. Each slot of the array is a variable that can hold a value and works like any variable. As with one-dimensional arrays, every slot in a 2D array is of the same type.

**Note:**

* Rows are numbered from 0 to N-1, where N is the number of rows
* Columns are numbered from 0 to M-1, where M is the number of columns.
* A 2D array with N rows and M columns will have N \* M slots.

**2. Declaring 2D-arrays**

In C the declaration of a 2D-array has the form:

**dataType arrayName[numRows][numColumns];**

where numRows and numColumns are non-negative integer expressions. An array declaration only reserves memory for the elements; it does not initialize the array.

Examples:

**int values[3][4];**

**double prices[10][5];**

**char code[6][26];**

**int rows, columns;**

**printf("Enter number of row and columns: "); // In C99 and above the**

**// dimensions of an array can be specified at run-time**

**scanf("%d%d", &rows, &columns);**

**double quizGrades[rows][columns];**

**3. Initializing 2D Array by an initializer-list**

A 2D-array can be initialized **row-wise** by an initializer list when it is declared:

int values[4][3] = {{8, 16, 9}, {3, 15, 7}, {2, 4, 5}, {11, 0, 3}};

the row-dimension and/or the inner curly braces may be removed in such a declaration:

int values[][3] = {8, 16, 9, 3, 15, 7, 2, 4, 5, 11, 0, 3};

The number of initializer may be less than the size of the array. Example:

**double grades[5][4] = {6.0};**

initializes grades[0][0] to 6.0 and each of the remaining 19 elements to 0.0

If the number of initializers is more than the number specified by the dimensions, an error occurs:

int values[4][3] = {{8, 16, 9}, {3, 15, 7}, {2, 4, 5}, {11, 0, 3}, {11, 0, 3}}; // Error

int x[4][3] = {{8, 16, 9, 6}, {3, 15, 7}, {2, 4, 5}, {11, 0, 3}}; // Error

**4. Accessing individual elements**

An individual element of a 2-D array can be accessed by specifying the index of row and column. For example

grades[2][4] = 95;

assigns the value 95 to the element in the third row and fifth column of array grades. The indexed (subscripted) variable can be used wherever an ordinary variable can be used.

**5. 2D array manipulation**

Example: Each of 4 students has taken 3 quizzes

double quiz[][3] = {{2.0, 1.5, 1.7}, {0.5, 1.0, 0.0}, {2.0, 2.0, 2.0}, {1.2, 0.7, 1.0}};

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 | 2.0 | 1.5 | 1.7 |
| 1 | 0.5 | 1.0 | 0.0 |
| 2 | 2.0 | 2.0 | 2.0 |
| 3 | 1.2 | 0.7 | 1.0 |

5.1 To manipulate a single row, fix the row index and use a single column loop

Example: Find the total quiz score for student with row index 3

**int c, r, numQuizzes = 3;**

**double sum = 0;**

**for(c = 0; c < numQuizzes; c++)**

**sum += quiz[3][c];**

**printf(**"**Total quiz score for student#3 = %.2f\n**"**, sum);**

5.2 To manipulate a single column, fix the column index and use a single row loop

Example: Find the average of quiz#0

**int c, r, numStudents = 4;**

**double sum = 0;**

**for(r = 0; r < numStudents; r++)**

**sum += quiz[r][0];**

**printf(**"**Quiz#0 average = %.2f\n**"**, sum / 4);**

5.3 To manipulate each row, use nested loops in which the row loop is the outer loop

Example: Find the total quiz score for each student

**int c, r, numStudents = 4, numQuizzes = 3;**

**double sum;**

**for(r = 0; r < numStudents; r++){**

**sum = 0;**

**for(c = 0; c < numQuizzes; c++){**

**sum += quiz[r][c];**

**}**

**printf(**"**Total quiz score for student#%d = %.2f\n**"**, r, sum);**

**}**

Example: Read the quiz array row-wise:

**int c, r, numStudents = 4, numQuizzes = 3;**

**double quiz[numStudents][numQuizzes];**

**for( r = 0; r < numStudents; r++ ){**

**for(c = 0; c < numQuizzes; c++){**

**scanf(**"**%lf**"**, &quiz[r][c]);**

**}**

**}**

**Suppose the input is:**

**5.0 2.0**

**3.0**

**4.0 1.0 3.5 2.5**

**1.5 6.0 5.5 7.0**

**8.0**

**The array quiz is initialized as:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 | 5.0 | 2.0 | 3.0 |
| 1 | 4.0 | 1.0 | 3.5 |
| 2 | 2.5 | 1.5 | 6.0 |
| 3 | 5.5 | 7.0 | 8.0 |

5.4 To manipulate each column, use nested loops in which the column loop is the outer loop

Example 01: Find the average of each quiz

**int c, r, numStudents = 4, numQuizzes = 3;**

**double sum;**

**for( c = 0; c < numQuizzes; c++ ){**

**sum = 0;**

**for( r = 0; r < numStudents; r++ ){**

**sum += quiz[r][c];**

**}**

**printf(**"**Quiz#%d average = %.2f\n**"**, c, sum / 4);**

**}**

Example 02: Read the quiz array column-wise:

**int c, r, numStudents = 4, numQuizzes = 3;**

**double quiz[numStudents][ numQuizzes];**

**for(c = 0; c < numQuizzes; c++){**

**for(r = 0; r < numStudents; r++){**

**scanf(**"**%lf**"**, &quiz[r][c]);**

**}**

**}**

**Suppose the input is:**

**5.0 2.0**

**3.0**

**4.0 1.0 3.5 2.5**

**1.5 6.0 5.5 7.0**

**8.0**

**The array quiz is initialized as:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 | 5.0 | 1.0 | 6.0 |
| 1 | 2.0 | 3.5 | 5.5 |
| 2 | 3.0 | 2.5 | 7.0 |
| 3 | 4.0 | 1.5 | 8.0 |

Note: There are problems that can be solved by manipulating a 2D-array either row-wise or column-wise:

Examples:

* Find a maximum or minimum element of a 2D-array.
* Find the sum or product of all elements in a 2D- array

**int c, r, numStudents = 4, numQuizzes = 3;**

**double sum = 0;**

**for( r = 0; r < numStudents; r++ ){**

**for(c = 0; c < numQuizzes; c++){**

**sum += quiz[r][c];**

**}**

**}**

**printf(**"**Sum of all quiz grades = %.2f\n**"**, sum);**

or:

**int c, r, numStudents = 4, numQuizzes = 3;**

**double sum = 0;**

**for(c = 0; c < numQuizzes; c++){**

**for(r = 0; r < numStudents; r++){**

**sum += quiz[r][c];**

**}**

**}**

**printf(**"**Sum of all quiz grades = %.2f\n**"**, sum);**

**6. Using 2-D arrays in functions**

There are three ways a 2D-array can be used with a function:

* Pass an individual element of the array. This can be done either call by value or call by reference.
* Pass one row of the array to a function.
  + This is done by indexing the array name with the row index. The corresponding formal parameter is a 1D-array declaration in which the array size may be omitted. The row is passed call by reference.
* Pass the entire 2D-array to a function
  + This is done by using the array name as actual argument. The corresponding formal parameter is a 2D-array declaration in which the row dimension may be omitted; but the column dimension cannot be omitted. The array is passed call by reference.

**Example 01:** Passing an individual 2D-array element to a function

|  |
| --- |
| **#include <stdio.h>**  **void test(int x, int\* y);**  **int main(void){**  **int array[3][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};**  **printf("Before the call to function test:\n");**  **printf("array[1][2] = %d\n", array[1][2]);**  **printf("array[2][3] = %d\n", array[2][3]);**    **test(array[1][2], &array[2][3]);**    **printf("\nAfter the call to function test:\n");**  **printf("array[1][2] = %d\n", array[1][2]);**  **printf("array[2][3] = %d\n", array[2][3]);**  **printf("\nThe element array[1][2] was passed call by value \n"**  **"and the element array[2][3] was passed call by reference\n");**  **return 0;**  **}**  **void test(int x, int\* y){**  **x = 40;**  **\*y = 60;**  **}** |
| Output: |

**Example 02:** Passing a row of a 2D-array to a function

|  |
| --- |
| **#include <stdio.h>**  **#define NUMROWS 3**  **#define NUMCOLUMNS 4**  **int sumRow(int x[]);**  **int main(void){**  **int array[NUMROWS][NUMCOLUMNS] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};**  **int sum = sumRow(array[0]); // pass row 0 to function**  **printf("Sum of row0 is %d\n", sum);**  **sum = sumRow(array[2]); // pass row 2 to function**  **printf("Sum of row2 is %d\n", sum);**  **return 0;**  **}**  **int sumRow(int x[]){**  **int k, rowSum = 0;**  **for(k = 0; k < NUMCOLUMNS; k++){**  **rowSum += x[k];**  **}**  **return rowSum;**  **}** |
| Output: |

**Example 03:** Passing an entire 2D-array to a function.

Write a C program that reads two matrices of maximum size 10\*10 it then adds the two matrices and displays the result of the addition:

|  |
| --- |
| **#include<stdio.h>**  **#define ROWS 10**  **#define COLS 10**  **void read\_2d\_array(int a[ ][COLS], int rows, int cols);**  **void add\_2d\_arrays(int a[ ][COLS], int b[ ][COLS], int c[ ][COLS], int rows, int cols);**  **void print\_2d\_array(int a[ ][COLS], int rows, int cols);**  **int main(void) { // for each function, rows and cols are input parameters:**  **int i, j, a[ROWS][COLS], b[ROWS][COLS], c[ROWS][COLS], rows, cols;**  **printf("Enter number of rows for the Matrix: ");**  **scanf("%d", &rows);**  **printf("Enter number of columns for the Matrix: ");**  **scanf("%d", &cols);**  **read\_2d\_array(a, rows, cols);**  **read\_2d\_array(b, rows, cols);**  **add\_2d\_arrays(a, b, c, rows, cols);**  **printf("The sum of two matrices is: \n");**  **print\_2d\_array(c, rows, cols);**  **return 0;**  **}**  **void read\_2d\_array(int a[ ][COLS], int rows, int cols) { // a is output parameter**  **int i, j;**  **printf("Enter the %d elements of the 2-D array row-wise: \n", rows \* cols);**  **for(i=0; i<rows; i++) {**  **for(j=0; j<cols; j++)**  **scanf("%d", &a[i][j]);**  **}**  **}**  **// a and b are input parameters, c is an output parameter:**  **void add\_2d\_arrays(int a[ ][COLS], int b[ ][COLS], int c[ ][COLS], int rows, int cols) {**  **int i, j;**  **for (i=0; i<rows; i++) {**  **for (j=0; j<cols; j++)**  **c[i][j] = a[i][j] + b[i][j];**  **}**  **}**  **void print\_2d\_array(int a[ ][COLS], int rows, int cols) { // a is an input parameter**  **int i, j;**  **for(i=0; i<rows; i++) {**  **for (j=0; j<cols; j++)**  **printf("%5d ", a[i][j]);**  **printf("\n");**  **}**  **}** |

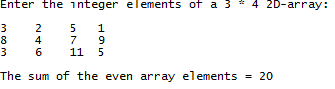
Laboratory Tasks

1.Write a C program that prompts for and reads a 3\*4 integer array **row-wise**. It then computes and displays

the sum of the even integers in the array. Use appropriate loops and only the main function in your

solution.

Sample program run:



2. Write a function **linearSearch2D** that takes a 2D int array and an integer value to be searched in the

array. The function returns the row and column indexes of the first element in the array that equals

to the search value (Note: search the array **column-wise**). If the search is not successful, the

function returns -1 and -1 for the row and column index.

Write a main function to test the **linearSearch2D** function. Use the dimensions 4 \* 3 for your

2D-array.

Sample program runs:

|  |
| --- |
|  |
|  |
|  |

3. The correct answers to a 5 question true/false test are: T T F F T. Write a C program that

initializes a 1D **char** array with the correct answers, and a 2D **char** array of size 6 \* 5 with the

following 6 student answers:

T F T T T

T T T T T

T T F F T

F T F F F

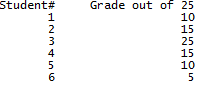
F F F F F

F T T T F

The program then passes these two arrays to a function **getGrades** that returns the grades of the six

students in a 1D int array. The main function finally displays the grades of the six students.

Assuming that each question is worth 5 points, the output, in the main function, is:



4. Each line of a text-file **grades.txt** contains a student ID and his grades in **5** quizzes:

|  |
| --- |
| 9001230 80.0 90.0 70.5 100.0 60.0  9001232 98.0 85.0 100.0 99.0 89.0  9001234 90.0 72.0 0.0 78.0 98.0  9001236 85.0 72.5 95.0 75.0 64.5  9001238 67.0 11.0 28.0 89.5 85.0 |

Given that the number of students is **5**, write a C program that reads the data from **grades.txt** into an integer 1D-array of student IDs and the corresponding quiz grades in a parallel 2D-array of type double. The program then calls the function **studentAverages** that receives the 2D array of grades and returns a 1D-array containing the average of each student. The main function finally prints an output in the format:

**ID Quiz Average**

9001230 80.1

9001232 94.2

9001234 67.6

9001236 78.4

9001238 56.1

Note:

* Your program must use appropriate loops.
* Your program must be general; it must work for any text-file with the above format.
* The **studentAverages** function must not contain **scanf** and **printf** statements.