Multi-Dimensional Arrays

Last updated 9/9/21

Multi-Dimensional Arrays

- 2 Dimensional Arrays

  Consider a table

1	2	3	4	5
6	5	4	3	2
12	11	13	14	15
19	17	16	3	1

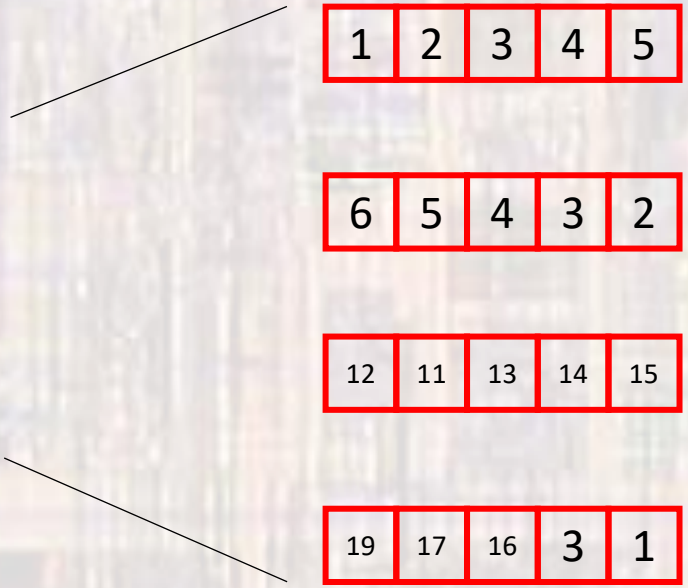
4 rows x 5 columns

Multi-Dimensional Arrays

- 2 Dimensional Arrays

  Consider a table

1	2	3	4	5
6	5	4	3	2
12	11	13	14	15
19	17	16	3	1

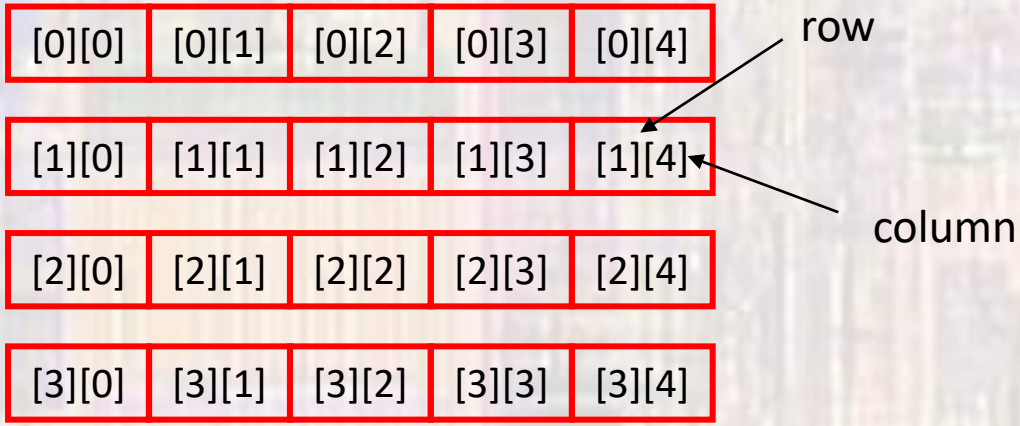


4 – 1 Dimensional Arrays

Multi-Dimensional Arrays

- 2 Dimensional Arrays

  Consider a table



  Array of Arrays – 4x5

  Indices are ROW-COL format

Multi-Dimensional Arrays

- 2 Dimensional Arrays

  Declaration

  ```
  type arrayName[#rows][#cols];
  ```

  Fixed size array – size known during compilation

  ```
  int scores[4][5];
  char first_name[15][20];
  ```

  Variable size array – size only known during execution

  ```
  float testAve[classSize][numTests];
  int numAs[gradesGE90][numClasses];
  ```

  where classSize,gradesGE90, numTests, numClasses are integral variables

Multi-Dimensional Arrays

- 2 Dimensional Arrays

  Initialization

  type arrayName[#rows][#cols] = {comma separated list};

```
int myArray[3][4] = {1,2,3,4,1,2,3,4,1,2,3,4};        // basic

int myArray[3][4] = {
                        {1,2,3,4},
                        {1,2,3,4},
                        {1,2,3,4}
                       };                              // preferred

int myArray[3][4] = {0};                               // all zeros
```

Multi-Dimensional Arrays

- 2 Dimensional Arrays

  Variable length arrays **cannot** have an initialization

```
float testAve[classSize][numTests];
int numAs[gradesGE90][numClasses];
```

Multi-Dimensional Arrays

- 2 Dimensional Arrays

  Accessing elements

```
foo = myArray[1][2];        // foo = 4
foo = myArray[2][foo];      // foo = 15
```

myArray

1	2	3	4	5
6	5	4	3	2
12	11	13	14	15
19	17	16	3	1

```
myArray[0][0] = 0;

foo = 1;
myArray[foo + 1][foo + 2] = 6;
```

0	2	3	4	5
6	5	4	3	2
12	11	13	6	15
19	17	16	3	1

Multi-Dimensional Arrays

- 2 Dimensional Arrays
  - Keyboard example
    - Read the 8 scores for 10 students from the keyboard and store them in a 2 dimensional array

```
int scores[10][8];
int row;
int col;
for(row = 0; row < 10; row++)
   for(col=0; col < 8; col++)
      scanf("%i", &scores[row][col]);
```

notes:

no {} since one line for each for

inner loop – columns (grades)
outer loop – rows (students)
reads all 8 scores for a student
then goes to the next student

&scores[row][col] refers to a single element (address)

Multi-Dimensional Arrays

- 2 Dimensional Arrays
  - Display example
    - Print the scores for 10 students from a 2 dimensional array to the console

```
int row;
int col;
for(row = 0; row < 10; row++){
   for(col=0; col < 8; col++)
      printf("%i", scores[row][col]);
   printf("\n");
}
```

notes:
inner loop – columns (grades)
outer loop – rows (students)
prints all 8 scores for a student
then goes to the next student

Multi-Dimensional Arrays

- 2 Dimensional Arrays
  - Assignment
    - Arrays must be copied element by element

```
int array1[10][8];
int array2[10][8];
…
int row;
int col;
for(row = 0; row < 10; row++)
   for(col=0; col < 8; col++)
      array2[row][col] = array1[row][col];
```

notes:
order does not matter
rows or col in outer loop

Multi-Dimensional Arrays

- Arrays in C
  - Example
    - Convert a 2D array to a 1D array

```
int array2D[10][8];
int array1D[80];
```

Multi-Dimensional Arrays

- Arrays in C
  - Example
    - Convert a 2D array to a 1D array

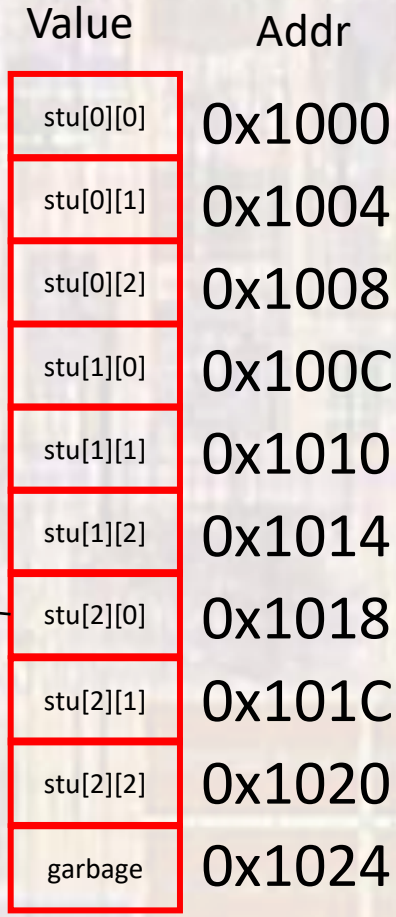
```
int array2D[10][8];
int array1D[80];
…
int row;
int col;
for(row = 0; row < 10; row++)
   for(col=0; col < 8; col++)
      array1D[row*8 + col] = array2D[ row][col];
```

notes:
order does matter
row must be in outer loop

Multi-Dimensional Arrays

- 2 Dimensional Arrays – Memory View
  - 3x3 array → linear in memory
  - C does NOT check array index ranges

```
int stu[3][3];

foo = stu[1][3];
    sets foo = stu[2][0] wrong

stu[3][2] = 12;
     overwrites critical data value
```



Multi-Dimensional Arrays

- Passing array values
  - Passing array values works just like any other value

```
fun1(myArray[3][7]);        // passes the value of myArray[3][7]
                            // to function fun1

fun2(&myArray[3][3]);       // passes a pointer to myArray
                            // element 3,3 (the address) to
                            // function fun2
```

Multi-Dimensional Arrays

- Passing array values
  - Passing the whole array
    - If we pass all the elements of a large array to multiple functions we use up a lot of data memory
    - Instead we pass the address of the array (by reference)
  - Remember – the name of the array is already an address to the beginning of the array
  - Must provide the 2nd dimension to compile

declaration

```
void fun3(int ary[ ][ val]);   // the array notation name[][#]
                               // tells the compiler it is expecting an
                               // address
```

…

call

```
fun3(myArray);                 // the array name is already an
                               // address
```

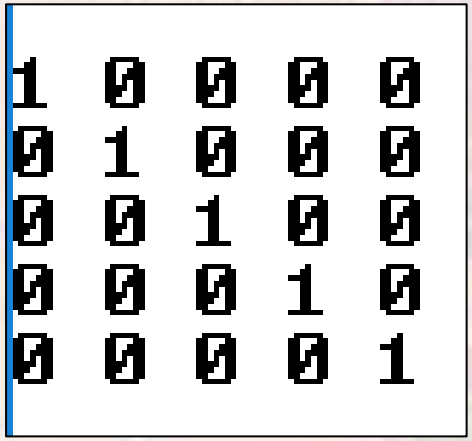
Multi-Dimensional Arrays

- Passing array values
  - Passing a ROW
    - We can pass just 1 row of 2-dimensional array to a function

```
int valArray[10][10];          // 2d array

declaration  void fun1d(int myArray[ ]);   // the array notation name[]
                                           // tells the compiler it is expecting an
                                           // address
                                           // only references a 1d array

call         fun1d(valArray[5]);           // passes only the row with index 5
```

Multi-Dimensional Arrays

- 2-Dimensional Array example
  - Create an identity matrix
  - 1s on the diagonal, 0 everywhere else



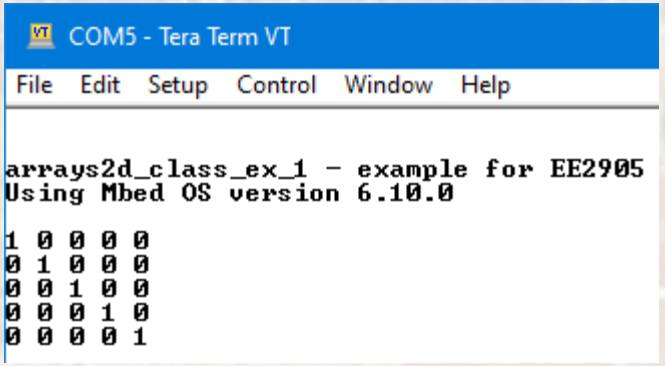
Multi-Dimensional Arrays

- 2-Dimensional Array example
  - Create an identity matrix

Note: passing the array and the dimensions
This function works for any size array

```
////////////////////////////////////
//
// arrays2d_class_ex_1 project
//
// created 5/12/21 by tj
// rev 0
//
////////////////////////////////////
//
// 2d array example file for class
//
// Create an identity matrix
//
////////////////////////////////////

#include "mbed.h"
#include <stdio.h>

#define ROW_NUM 5
#define COL_NUM 5

// Function Prototypes (Declarations)
void print_array_2d(int num_rows, int num_cols, const int the_ary[][COL_NUM]);

int main(void){
    setbuf(stdout, NULL);   // fix for terminal issue

    // splash
    printf("\n\narrays2d_class_ex_1 - example for EE2905\n");
    printf("Using Mbed OS version %d.%d.%d\n\n",
           MBED_MAJOR_VERSION, MBED_MINOR_VERSION, MBED_PATCH_VERSION);

    // local variables
    int my_array[ROW_NUM][COL_NUM];
    int row;
    int col;

    // create the identiy matrix
    for(row = 0; row < ROW_NUM; row++){
        for(col = 0; col < COL_NUM; col++){
            if(row == col)
                my_array[row][col] = 1;
            else
                my_array[row][col] = 0;
        } // end for - col
    } // end for - row

    print_array_2d(ROW_NUM, COL_NUM, my_array);

    return 0;
}// end main
```

```
// Function Definitions
void print_array_2d(int num_rows, int num_cols, const int the_ary[][COL_NUM]){
    // print 2d array

    // local variables
    int row;
    int col;

    // print matrix
    for(row = 0; row < num_rows; row++){
        for(col = 0; col < num_cols; col++){
            printf("%d ", the_ary[row][col]);
        } // end for - col
    printf("\n");
    } // end for - row

    return;
}// end print_array_2d
```



Multi-Dimensional Arrays

- 2-Dimensional Array example
  - Create an identity matrix

```
//////////////////////////////////////
//
// arrays2d_class_ex_1 project
//
// created 5/12/21 by tj
// rev 0
//
//////////////////////////////////////
//
// 2d array example file for class
//
// Create an identity matrix
//
//////////////////////////////////////

#include "mbed.h"
#include <stdio.h>

#define ROW_NUM 5
#define COL_NUM 5

// Function Prototypes (Declarations)
void print_array_2d(int num_rows, int num_cols, const int the_ary[][COL_NUM]);

int main(void){
    setbuf(stdout, NULL);   // fix for terminal issue

    // splash
    printf("\n\narrays2d_class_ex_1 - example for EE2905\n");
    printf("Using Mbed OS version %d.%d.%d\n\n",
           MBED_MAJOR_VERSION, MBED_MINOR_VERSION, MBED_PATCH_VERSION);

    // local variables
    int my_array[ROW_NUM][COL_NUM];
    int row;
    int col;

    // create the identiy matrix
    for(row = 0; row < ROW_NUM; row++){
        for(col = 0; col < COL_NUM; col++){
            if(row == col)
                my_array[row][col] = 1;
            else
                my_array[row][col] = 0;
        } // end for - col
    } // end for - row

    print_array_2d(ROW_NUM, COL_NUM, my_array);

    return 0;
}// end main
```

```
// Function Definitions
void print_array_2d(int num_rows, int num_cols, const int the_ary[][COL_NUM]){
    // print 2d array

    // local variables
    int row;
    int col;

    // print matrix
    for(row = 0; row < num_rows; row++){
        for(col = 0; col < num_cols; col++){
            printf("%d ", the_ary[row][col]);
        } // end for - col
    printf("\n");
    } // end for - row

    return;
}// end print_array_2d
```

Note: Constant 2nd dimension

Row size could have been left as a variable

COM5 - Tera Term VT

File Edit Setup Control Window Help

```
arrays2d_class_ex_1 - example for EE2905
Using Mbed OS version 6.10.0

1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
0 0 0 1 0
0 0 0 0 1
```

Multi-Dimensional Arrays

- N Dimensional Arrays
  - No limit to how many dimensions our array can be
  - Syntax follows 2-D approach
  - must provide value for all dimensions beyond the 1st

```
int myArray[3][3][3];        // Rubiks Cube

float myArray[12][3][7][2][100];

fun1(myArray[6][2][3]);
…
int fun1(float theArray[ ][valy][valz]){
…
```

Constant valy, valz

Multi-Dimensional Arrays

- N Dimensional Arrays
  - Can provide the additional dimensions in the call

```
float myArray[6][2][3];
…
fun1(2, 3, myArray);
…
int fun1(int y, int z, float theArray[ ][y][z]){
…
```

Not in mbed