Multi-Dimensional Arrays and Functions

Last updated 6/19/23

These slides introduce using multi-dimensional arrays in functions

Multi-Dim Arrays and Functions

- 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-Dim Arrays and Functions

- Passing the whole array to a function
  - 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 ALL dimensions beyond the first
    - Required to calculate the offsets for the indices

function declaration

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

call

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

C does not have a way to pass a copy of an array to a function

Multi-Dim Arrays and Functions

- Accessing the array inside a function
  - We passed a pointer to the array into the function

```
void fun3(int ary[ ][3]);        // function expecting a pointer
…
fun3(myArray);                   // pass a pointer to the function
```

  - Inside the function, ary has the value of the pointer
    - This is the address of the array
  - To access an element of the array we can use the normal array notation since the name we are using is already a pointer
    - This is the normal array situation

inside the function

```
foo = myArray[3][2];

myArray[2][0] = 5;

scanf("%i", &myArray[7][1]);
```

Multi-Dim Arrays and Functions

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

```
int valArray[10][10];
fun1d(valArray[5]);              // passes only the row with index 5

void fun1d(int myArray[ ]);      // the array notation name[]
                                 // tells the compiler it is expecting an
                                 // address

                                 // only references a 1d array
```

Multi-Dim Arrays and Functions

- Passing array values
  - What if we want to pass the whole array to a function but we do not want the function to modify the array?
  - Declare the array as a constant in the function declaration and definition

```
float average(int myArray[ ][3]);          // modifiable
```

→

```
float average(const int myArray[ ][3]);    // non-modifiable
```

Multi-Dim Arrays and Functions

- 2-Dimensional Array example
  - Create an identity matrix and then print it

```c
/*
 * array_examples_2d.c
 *
 *  Created on: Jan 23, 2018
 *      Author: johnsontimoj
 */

#include <stdio.h>

#define row_num 5
#define col_num 5

// function prototypes
void print_array_2d(int num_row, int num_col, const int the_array[][col_num]);

int main(void){
    setbuf(stdout, NULL);  // disable buffering

    // local variables
    int my_array[row_num][col_num];
    int row;
    int col;

    // create identity 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 of inner for

    print_array_2d(row_num, col_num, my_array);

    return 0;
} // end main
```

```c
void print_array_2d(int num_row, int num_col, const int the_array[][col_num]){
    int row;
    int col;
    for(row = 0; row < num_row; row++){
       for(col = 0; col < num_col; col++)
           printf("%d ", the_array[row][col]);
       printf("\n");
    } // end of for

    return;
}// end print_array_2d
```

```
<terminated> (exit value: 0) C
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-Dim Arrays and Functions

- 2-Dimensional Array example
  - Create an identity matrix

```c
/*
 * array_examples_2d.c
 *
 *  Created on: Jan 23, 2018
 *      Author: johnsontimoj
 */

#include <stdio.h>

#define row_num 5
#define col_num 5

// function prototypes
void print_array_2d(int num_row, int num_col, const int the_array[][col_num]);

int main(void){
    setbuf(stdout, NULL);  // disable buffering

    // local variables
    int my_array[row_num][col_num];
    int row;
    int col;

    // create identity 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 of inner for

    print_array_2d(row_num, col_num, my_array);

    return 0;
} // end main
```

```c
void print_array_2d(int num_row, int num_col, const int the_array[][col_num]){
    int row;
    int col;
    for(row = 0; row < num_row; row++){
       for(col = 0; col < num_col; col++)
           printf("%d ", the_array[row][col]);
       printf("\n");
    } // end of for

    return;
}// end print_array_2d
```

Note: Constant 2nd dimension

```
<terminated> (exit value: 0) C
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-Dim Arrays and Functions

- N > 2 Dimensional Arrays
  - All dimensions except the first must be provided

Function Declaration

```
int fun1(int dim1, float theArray[ ][3][5][9]);
```

```
…
float myArray[6][3][5][9];
…
```

Call

```
fun1(6, myArray);
…
```

Function Definition

```
int fun1(int dim1, float theArray[ ][3][5][9]){
   …
}
```

Multi-Dim Arrays and Functions

- N > 1 Dimensional Arrays
  - Can provide the additional dimensions in the call

Function Declaration

```
int fun1(int x, int y, int z, float theArray[ ][y][z]);
```

```
…
float myArray[6][2][3];
…
```

Call

```
fun1(6, 2, 3, myArray);
```

```
…
```

Function Definition

```
int fun1(int x, int y, int z, float theArray[ ][y][z]){
…
```