C Chapter 6: Arrays

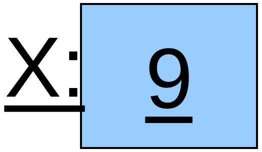
CECS130
Introduction to Programming Languages
Dr. Roman V. Yampolskiy

Intro

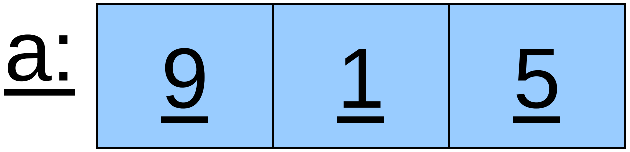
- So far we have used variables that can store and hold only one value
- These variables are called ***atomic*** since they cannot be decomposed into more basic values
- Sometimes we are faced with a situation where we have a group of values of the same type that are logically related such as a set of grades for a student
- Computers allow us to store these groups together in what we call "***arrays***".

Arrays

- C/C++ constructs that allow one to associate multiple data items with one variable
- One variable stores more than one data item
- Variable storing single integer:

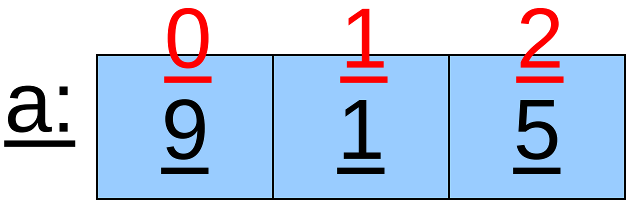

X: 9

- Variable storing three integers:


a: 9 1 5

Arrays

- How do we access each item in the variable; i.e. how do we reference any specific compartment?



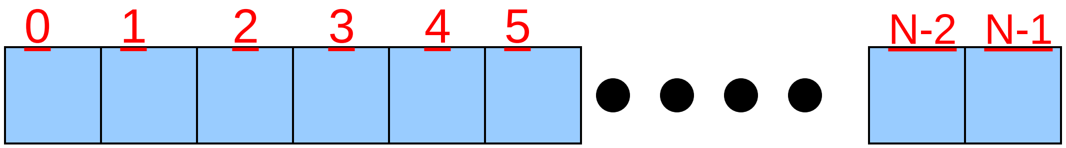
	0	1	2
a:	9	1	5

- Each cell has a number known as index
- Index specifies the compartment of the variable

- The proper name of the compartment is the ***array cell***

Arrays (one– dimensional)

- List of related values stored in a structure where cells are arranged in *one row* and *multiple columns*.
- A typical one-dimensional array may be viewed as:



- IMPORTANT: Array indexing is ***zero-based***
  - First cell always has an index *0*.
  - In the array of *N* cells, last cell always has an index *N-1*.
  - An attempt to use the index greater than N-1 or less than 0 to access a cell in the array of N cells, will result in an error
    - The value retrieved may be unpredictable
    - Program may crash

Arrays (one– dimensional)

- When would we need to use arrays?
  - To store multiple related values
    - Test scores
    - Student records
    - Experimental results

- Declaring array variables:

  type name[size];

  - *type* indicates the data type of the values to be stored in the array
  - *name* is the name of the array variable
  - *size* is the size of the array variable
    - *Note:* when declaring array of *N* elements, the value of size is equal to *N*

Arrays (one– dimensional)

- Array cells are accessed by specifying array name and the index in square brackets:

  name[*index*]

- Analogy:
  - If you live in a house you only specify the house number for your address
    - That's equivalent to using regular variable
  - If you live in an apartment building, you specify the building and the apartment number in address

- *Because arrays are zero-based, the value of index to access i^{th} cell is always i – 1.*
  - *Index of first cell is 0*
  - *Index of second cell is 1*
  - *Index of i^{th} cell is i–1.*

Array Initialization

- Arrays may be initialized by initializing their individual elements by
  - using several assignment statements (the long way)
  - or by using a loop structure (the quick way)
- Arrays may also be initialized by setting the array equal to a set of values separated by commas
  - The set of values should not exceed the size of the array
- C does not perform bounds check
  - It does not guard against putting data past the end of the array
  - You must be careful not to do that

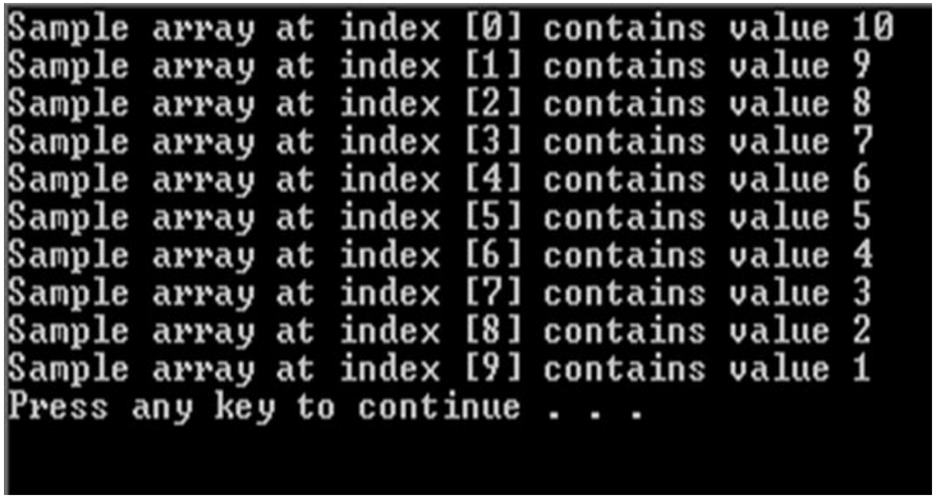
Array Initialization within Declaration

- int iArray[5] = {0, 1, 2, 3, 4};
- int iArray[5] = {0};   //single default value
- int crt[] = {6, 7, 8};
- How about int A[4] = {3,4}; ?
  - A[0] = 3, A[1] = 4, A[2] = 0, A[3] = 0

- Character Arrays store special null termination character (more on this in chapter 8: Strings)
  - char cName[] = {‘O’, ‘l’, ‘i’, ‘v’, ‘i’, ‘a’, ‘\0’};
  - char cName[] = “Olivia”;
    - Array holds 7 characters 6 in Olivia + ‘\0’
    - Size of char arrays needs to accommodate \0

Array Example: Initialization with a Loop

```
#include <stdio.h>

int main() {

    int sample[10];        // Declare array to hold 10 integers
    int index;             // Array index
    int values = 10;

    //Put a value into every cell of the array
    for (index = 0; index < 10; index++)
    {        // initialize array
             sample[index] = values;
             values--;
    }

    //Print out values stored in array
     for (index = 0; index < 10; index++)
    {
             printf("Sample array at index [%d] contains value %d \n", index, sample[index]);
    }                                                                      //access array cell
    system("pause");
    return 0;
}
```



```
Sample array at index [0] contains value 10
Sample array at index [1] contains value 9
Sample array at index [2] contains value 8
Sample array at index [3] contains value 7
Sample array at index [4] contains value 6
Sample array at index [5] contains value 5
Sample array at index [6] contains value 4
Sample array at index [7] contains value 3
Sample array at index [8] contains value 2
Sample array at index [9] contains value 1
Press any key to continue . . .
```

Arrays (one– dimensional)

- In C/C++ arrays consist of contiguous memory locations
- All array elements reside next to each other in memory
- Lowest address corresponds to the first element
- Highest address corresponds to the highest element
- Array declared in previous example, after the execution of the first for loop looks like this:

sample[0]	sample[1]	sample[2]	sample[3]	sample[4]	sample[5]	sample[6]	sample[7]	sample[8]	sample[9]
10	9	8	7	6	5	4	3	2	1

Array Initialized by User: Example

```
#include <stdio.h>
int main() {

      int arrOfInts[10];           // Declare array of size 10
      int i;                      // Declare array index

      //Reading values into array (Exactly 10 values)
      for (i = 0; i < 10; i++) {
         printf("Enter value # %d\n", i + 1);
         scanf("%d", &arrOfInts[i]);
      }

      //Print out values
      for (i = 0; i < 10; i++)
      {
         printf("Value stored at cell %d  is %d\n", i, arrOfInts[i]);
      }
      system("pause");
      return 0;
}
```

Arrays (one– dimensional)

- Arrays are common in programming because they allow us to deal with a set of related values as with one variable
- Arrays allow us to keep related values together
- Processing values individually is accomplished with the use of loops
- Using loop iterations we can:
  - Search arrays for specific values
  - Calculate sums of elements
  - Count elements meeting specific criteria

Searching Arrays

- Searching arrays involves using loops
  - Counting loops are used, with counter variable used as the array *index*
  - Each iteration of the loop increments or decrements the *index*. This causes each cell to be examined individually
  - Loop is terminated by one of the two conditions:
    - Every element in the array has been examined
      - Value of *index* has exceeded the array bound
    - The cell that contains the value that satisfies the specified criteria has been found

Searching Arrays

- To search, we need to know the beginning and end of search.
  - How much of the array is used?
  - Which cells are relevant to the search?
- Example:
  - Given array of integers `arrOfInts[]`
  - Assume `arrOfInts[]` of size 10
  - Given integer value stored in variable `nVal`.
  - Find the *first* occurrence of the value stored in `nVal` in the array `arrOfInts[]`

Searching Arrays

- Following elements are needed
  - Index – variable to hold index of array cells
  - Maximum number of cells searched – variable to indicate a maximum number of array cells to be searched
  - Value sought – value we are trying to find in the array
  - Location – variable to which index of the location of the array that contains the value sought is assigned
  - Loop that iterates over every cell of the array
    - Index is incremented for every iteration
    - Looping continues as long as:
      1. Index has not exceeded the value of the Maximum number of cells searched variable.
         1. ALWAYS test for the index exceeding the value of Maximum number of cells searched variable first.
      2. Contents of the array cell currently indicated by index do not satisfy the Value sought

Searching Arrays: Example

```
#include <stdio.h>                    //Find the first occurrence of the value stored in nVal in the array arrOfInts[]
int main() {
        int arrOfInts[10];       // array is of size 10
        arrOfInts[0] = 20; arrOfInts[1] = 15; arrOfInts[2] = 1; arrOfInts[3] = 7;arrOfInts[4] = 17; arrOfInts[5] = 19; arrOfInts[6] = 10;
        arrOfInts[7] = 5; arrOfInts[8] = 0; arrOfInts[9] = 3;
        int nVal; //Declare a variable where the value sought is to be stored

        printf("Enter the value sought in the array: "); //Prompt the user for the value sought and start search
        scanf("%d", &nVal);
        int i = 0;                 // Declare and initialize array index
        int nLocation;             // Location of value sought
        //Search the array, comparing every cell to the value sought
        while ((i < 10) && (nVal != arrOfInts[i])) {
            i = i + 1;
        }
        if (i >= 10) {//See what stopped the loop. ALWAYS CHECK FOR INDEX BEING OUT OF BOUNDS FIRST
            //If index is out of bounds, the value has not been found
            nLocation = -1;   //Pick a value that cannot possible be an index
        } else { //Index still within bounds, that means there was a match
            nLocation = i;
        }
        //Print out results
        if (nLocation == -1) {
            printf("The value you provided was not found.\n\n");
        } else {
            printf("The value %d is located at array cell index %d \n", nVal, i);
        }
        system("pause"); return 0;
}
```

Counting Elements in Arrays

- Counting elements in arrays involves using loops.
  - Counting loops are used, with counter variable used as the array *index*
  - Each iteration of the loop increments or decrements the *index*. This causes each cell to be examined individually
  - In this case every cell of the array is examined
  - Unlike in the search procedure, iteration doesn’t stop when a match occurs. Here, we simply count matches
  - The loop is terminated when the index exceeds the maximum allowable value

Counting Elements in Arrays

- Following elements are needed
  - Index – variable to hold index of array cells
  - Number of cells checked– variable to indicate a number of array cells to be checked
  - Criterion – value to be matched
  - Match Counter – counter of the number of times the criterion was satisfied
  - Loop that iterates over every cell of the array

Iterating over Arrays: Counting Example

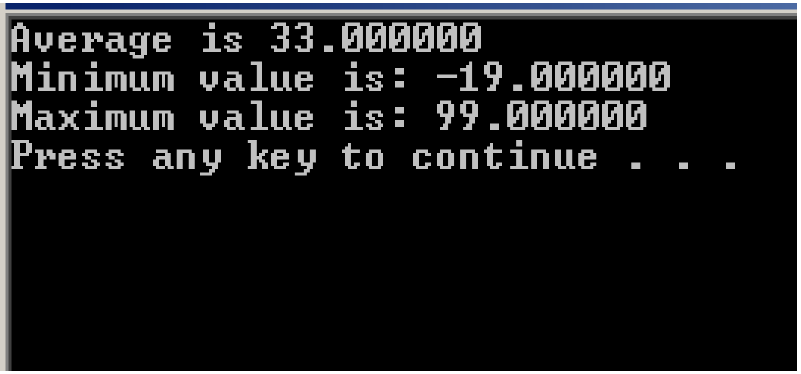
```
#include <stdio.h>
int main() {
        //Declare array of integers and initialize every cell to some random value.
        int arrOfInts[10];           // array is of size 10
        arrOfInts[0] = 20;arrOfInts[1] = 15;arrOfInts[2] = 10; arrOfInts[3] =17;
        arrOfInts[4] = 17; arrOfInts[5] = 19; arrOfInts[6] = 10; arrOfInts[7] = 5;
        arrOfInts[8] = 10; arrOfInts[9] = 20;
        int nVal;  //Declare a variable where the value representing count criterion

        printf("Enter the value of which occurrences are to be counted in the array: ");
        scanf("%d", &nVal);
        int i;                       // Declare array index
        int nCount = 0;              // Number of matches

        //Iterate over the entire array
        for (i = 0; i < 10; i++)
        {
           if (arrOfInts[i] == nVal)//Check if the current array cell matches
                   nCount++;
        }
        printf("The value %d occurs in the array %d times\n", nVal, nCount);
        system("pause");
        return 0;
}
```

Iterating over Arrays: Find Average

```
#include <stdio.h>

int main() {
      //Declaration and initialization of array of 10 integers
      int arrOfInts[] = {20, 10, 5, 20, 20, 10, 5, 5, 10, 15 };

      int i;                       // Declare array index
      int nAccum = 0;              // Count of matches
      float nAverage;                // Average value

      //Iterate over the entire array
      for (i = 0; i < 10; i++){
         nAccum = nAccum + arrOfInts[i];
      }
      nAverage = nAccum / 10;                // Calculate average

      //Print out results
      printf("The average is %f \n", nAverage);

      system("pause");
      return 0;
}
```

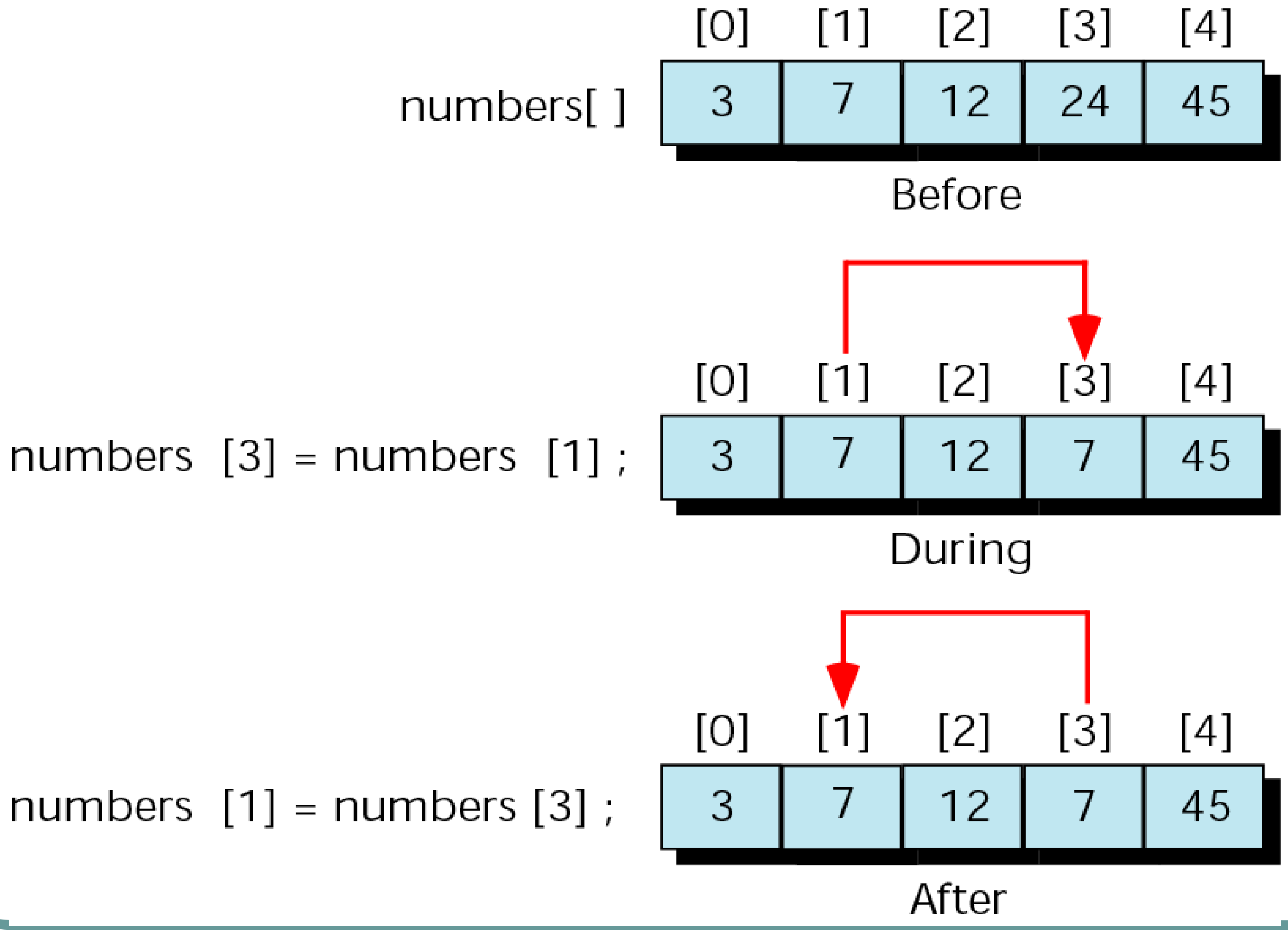
Arrays find Min and Max: Example

```c
#include <stdio.h>
int main(){
     float avg, min_val, max_val; float nums[10];
     nums[0] = 10; nums[1] = 18; nums[2] = 75; nums[3] = 0; nums[4] = 1; nums[5] = 56; nums[6] = 99; nums[7] = 12;
     nums[8] = -19; nums[9] = 88;

     avg = 0; int i;                    //Compute the average
     for (i = 1; i < 10; i++) {
             avg = avg + nums[i];
     }
     avg = avg / 10;
     printf("Average is %f\n", avg);

     //Find minimum and maximum values stored in the array.
     // 1. find the index
     // 2. use that index to access the value
     min_val = nums[0];    max_val = nums[0];
     int j;
     for (j = 1; j < 10; j++)
     {
             if (nums[j] < min_val)
                          min_val = nums[j];
             if (nums[j] > max_val)
                          max_val = nums[j];
     }
     printf("Minimum value is: %f\n", min_val);
     printf("Maximum value is: %f\n", max_val);
     system("pause");
     return 0;
}
```

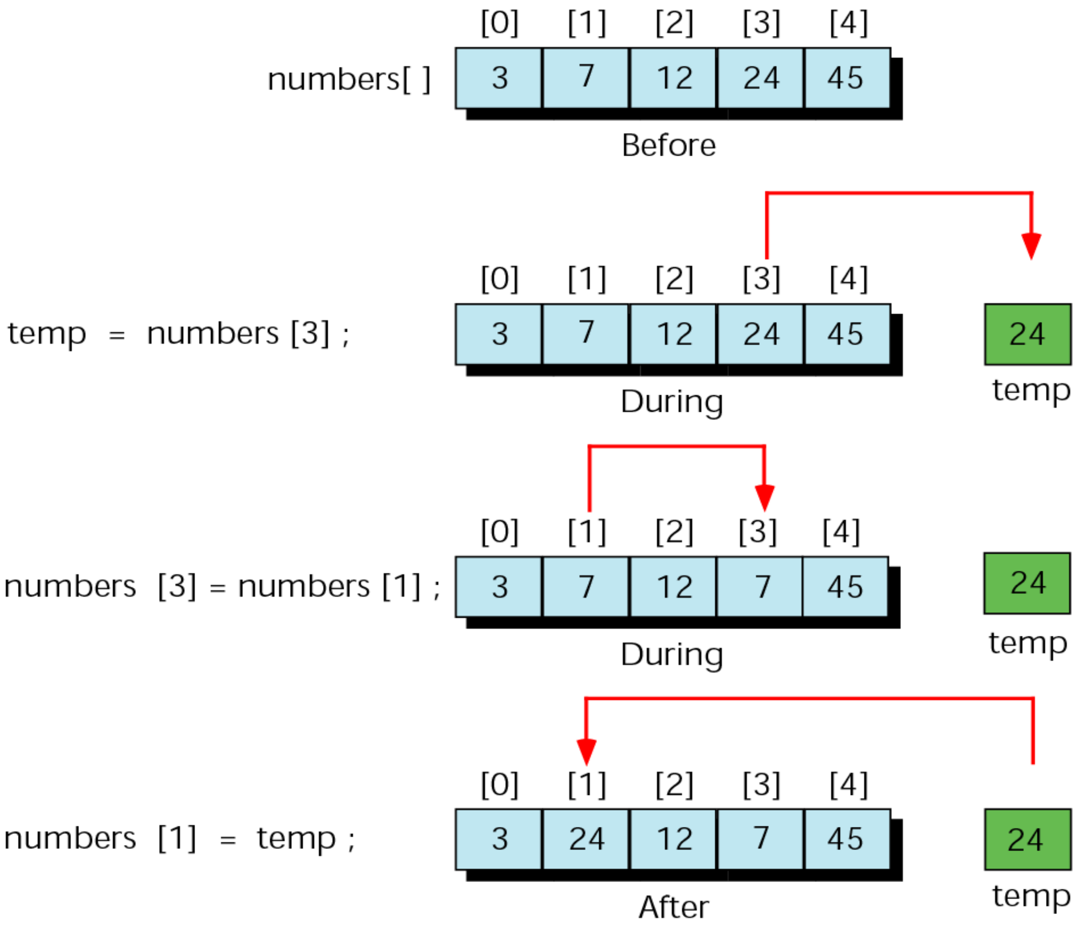


```
Average is 33.000000
Minimum value is: -19.000000
Maximum value is: 99.000000
Press any key to continue . . .
```

Exchanging Values: the Wrong Way



Exchanging Values: the Right Way



numbers[ ]

[0]	[1]	[2]	[3]	[4]
3	7	12	24	45

Before

```
temp  =  numbers [3] ;
```

[0]	[1]	[2]	[3]	[4]
3	7	12	24	45

During

temp
24

```
numbers  [3] = numbers [1] ;
```

[0]	[1]	[2]	[3]	[4]
3	7	12	7	45

During

temp
24

```
numbers  [1]  =  temp ;
```

[0]	[1]	[2]	[3]	[4]
3	24	12	7	45

After

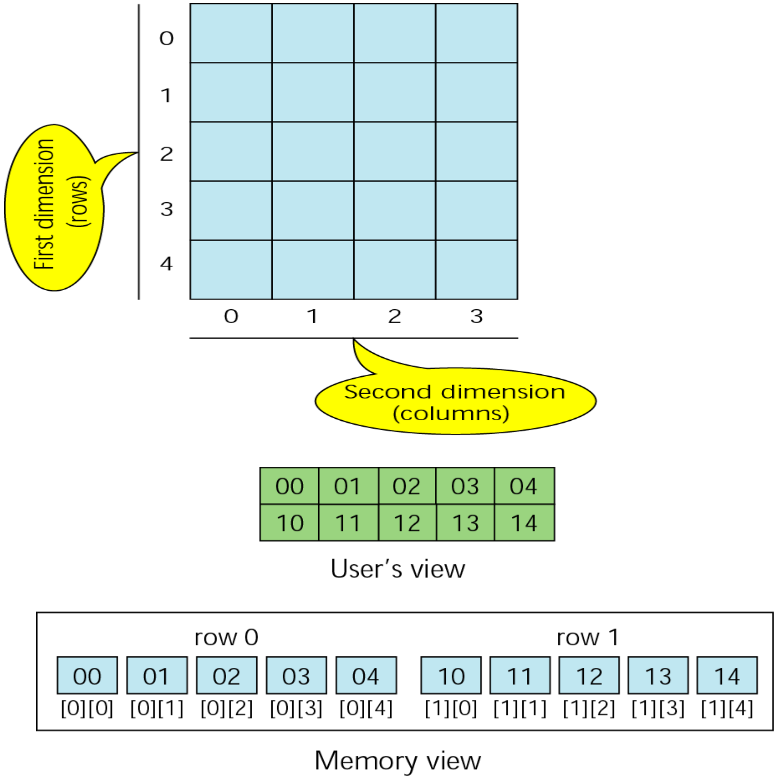
temp
24

Passing 1D Arrays to Functions

```
#include <stdio.h>
int addNumbers(int fiveNumbers[]); //declare function

int main() {
  int array[5];
  int i;
  printf("Enter 5 integers separated by spaces: ");
  for(i=0 ; i<5 ; i++) {
    scanf("%d", &array[i]);
  }
  printf("\nTheir sum is: %d\n", addNumbers(array));
  system("pause");
  return 0;
}
int addNumbers(int fiveNumbers[]) { /*define function */
  int sum = 0;
  int i;
  for(i=0 ; i<5 ; i++) {
    sum+=fiveNumbers[i];            /* work out the total */
  }
  return sum;                    /* return the total   */
}
```

- **The size of the array is blank in both the function declaration and definition - the compiler works it out for you**
- **The array isn't actually passed to the function - just the array's location in the memory**

2D Arrays



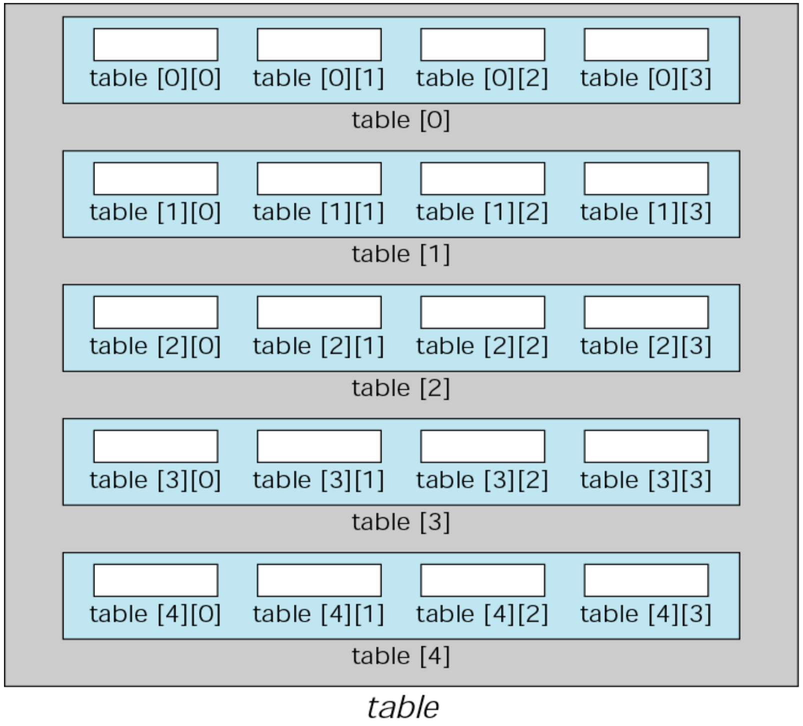


table [0][0] table [0][1] table [0][2] table [0][3]
table [0]
table [1][0] table [1][1] table [1][2] table [1][3]
table [1]
table [2][0] table [2][1] table [2][2] table [2][3]
table [2]
table [3][0] table [3][1] table [3][2] table [3][3]
table [3]
table [4][0] table [4][1] table [4][2] table [4][3]
table [4]
table

```
int array[4][3] =
   {1, 2, 3},
   {4, 5, 6},
   {7, 8, 9},
   {10, 11, 12}
};
```

(a)

Equivalent

```
int array[4][3];
array[0][0] = 1; array[0][1] = 2; array[0][2] = 3;
array[1][0] = 4; array[1][1] = 5; array[1][2] = 6;
array[2][0] = 7; array[2][1] = 8; array[2][2] = 9;
array[3][0] = 10; array[3][1] = 11; array[3][2] = 12;
```

(b)

Two-Dimensional Arrays

- Two-dimensional Array: a collection of a fixed number of components arranged in two dimensions
  - All components are of the same type
- The syntax for declaring a two-dimensional array is:

```
dataType arrayName[intexp1][intexp2];
```

  where `intexp1` and `intexp2` are expressions yielding positive integer values

- The two expressions `intexp1` and `intexp2` specify the number of rows and the number of columns, respectively, in the array
- Two-dimensional arrays are sometimes called matrices or tables

Initialization of 2D Arrays

- Like one-dimensional arrays
  - Two-dimensional arrays can be initialized when they are declared
- To initialize a two-dimensional array when it is declared
  1. Elements of each row are enclosed within braces and separated by commas
  2. All rows are enclosed within braces
  3. For number arrays, if all components of a row are not specified, the unspecified components are initialized to zero

2D Array Initialization Options

```
int first[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};

int second[3][4] = {0, 1, 2, 3,
                    4, 5, 6, 7,
                    8, 9,10,11};
                    /* a clearer definition than the first */

int third[][5] = {0,1,2,3,4};
                  /* third[] only has one index of 0 */

int fourth[][6] = {0,1,2,3,4,5,6,7,8,9,10,11};
                   /* fourth[] has 2 indices - 0 or 1 */

int fifth[][4] = {{0,1,2},{3,4,5},{6,7,8},{9,10,11}};
```

Processing Two-Dimensional Arrays

- A two-dimensional array can be processed in three different ways:

  1. Process the entire array
  2. Process a particular row of the array, called row processing
  3. Process a particular column of the array, called column processing

- Each row and each column of a two-dimensional array is a one-dimensional array

- When processing a particular row or column of a two-dimensional array

  - we use algorithms similar to processing one-dimensional arrays

Passing 2D Arrays to Functions

- Two-dimensional arrays can be passed as parameters to a function
- By default, arrays are passed by reference
- The base address, which is the address of the first component of the actual parameter, is passed to the formal parameter
- Two-dimensional arrays are stored in row order
    - The first row is stored first, followed by the second row, followed by the third row and so on
- When declaring a two-dimensional array as a formal parameter
    - can omit size of first dimension, but not the second
- Number of columns must be specified

Passing 2D Arrays to Functions

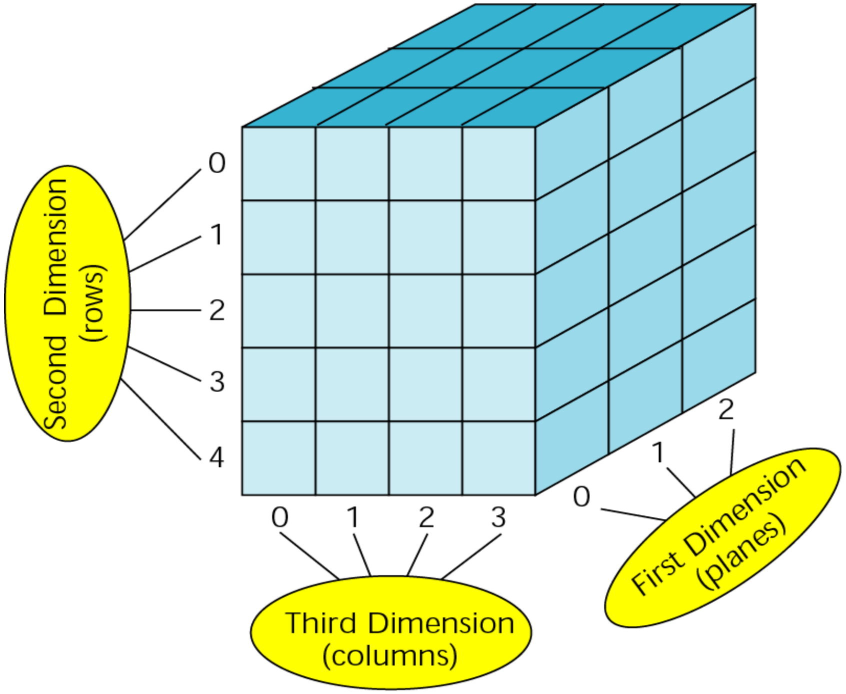
- This is similar to passing 1D arrays but, in the function declarations you must specify all the dimension sizes (but the leftmost one is optional).
- Use nested loops to work with 2D arrays

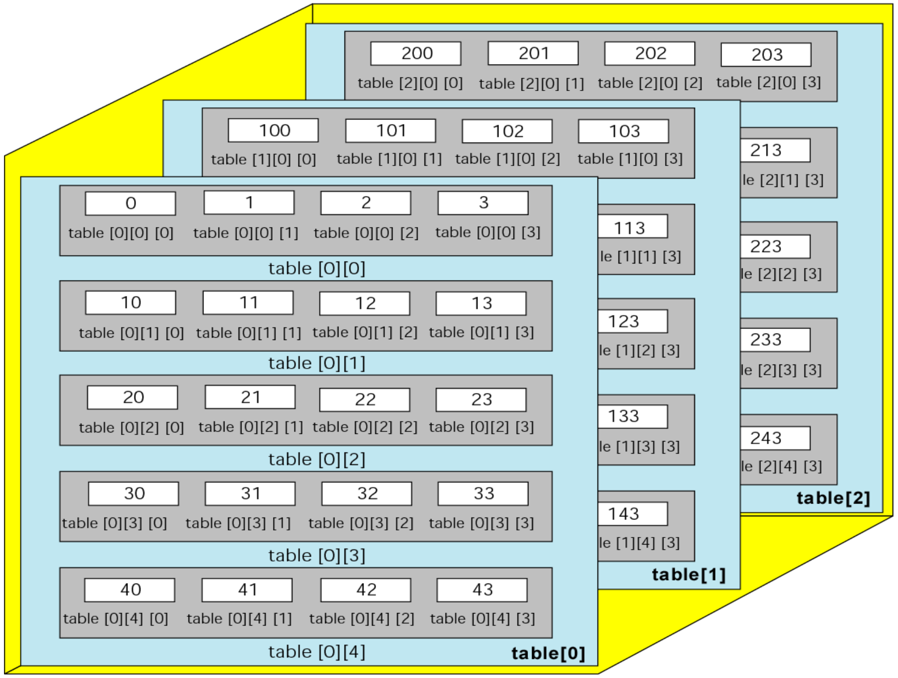
```
#include <stdio.h>

void printArray(int array[][4]); /* declare function */

int main() {
  int array[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
  printArray(array);
  system("pause");
  return 0;
}

void printArray(int array[][4]) { /* define function */
  int i, j;

  for(i=0 ; i<3 ; i++) {
    for(j=0 ; j<4 ; j++) {
      printf("%2d ", array[i][j]);
    }
    printf("\n");
  }
}
```

Scope of Arrays

- Arrays may be declared inside or outside functions.
- Arrays declared inside a function are *local arrays* and only accessible to the declaring function
- Arrays declared outside a function are *global arrays*
- *Global and local static arrays* are created once and retain their values until main() exits.
- *auto arrays* are created and destroyed each time the function defining them is called

Multidimensional Arrays





Multidimensional Arrays

- Multidimensional Array: collection of a fixed number of elements (called components) arranged in n dimensions (n >= 1)
- Also called an n-dimensional array
- General syntax of declaring an n-dimensional array is:

```
dataType arrayName[intExp1][intExp2]...[intExpn];
```

where `intExp1`, `intExp2`, … are constant expressions yielding positive integer values

Multidimensional Arrays (continued)

- The syntax for accessing a component of an `n`-dimensional array is:

  ```
  arrayName[indexExp1][indexExp2]...[indexExpn]
  ```

  where `indexExp1,indexExp2`,..., and indexExpn are expressions yielding nonnegative integer values
- `indexExpi` gives the position of the array component in the `i`th dimension

Multidimensional Arrays (continued)

- When declaring a multi-dimensional array as a formal parameter in a function
  - can omit size of first dimension but not other dimensions
- As parameters, multi-dimensional arrays are passed by reference only
- There is no check if the array indices are within bounds

The End!