CSE1142 – Arrays in C

Sanem Arslan Yılmaz

Some of the slides are from:
CMPE150 – Boğaziçi University
Deitel & Associates

Agenda

- Arrays
  - Definitions & Syntax
  - Initialization
  - Index and bounds
- Size of Arrays
- Examples
- Passing Arrays to Functions
- Sorting Arrays
- Searching Arrays
- Multidimensional Arrays

Motivation

- You may need to define many variables of the same type.
  - Defining so many variables one by one is cumbersome.

- Probably you would like to execute similar statements on these variables.
  - You wouldn't want to write the same statements over and over for each variable.

Arrays – Definitions & Syntax

- An array is a variable that is a collection of multiple values of the same type.
- Syntax:

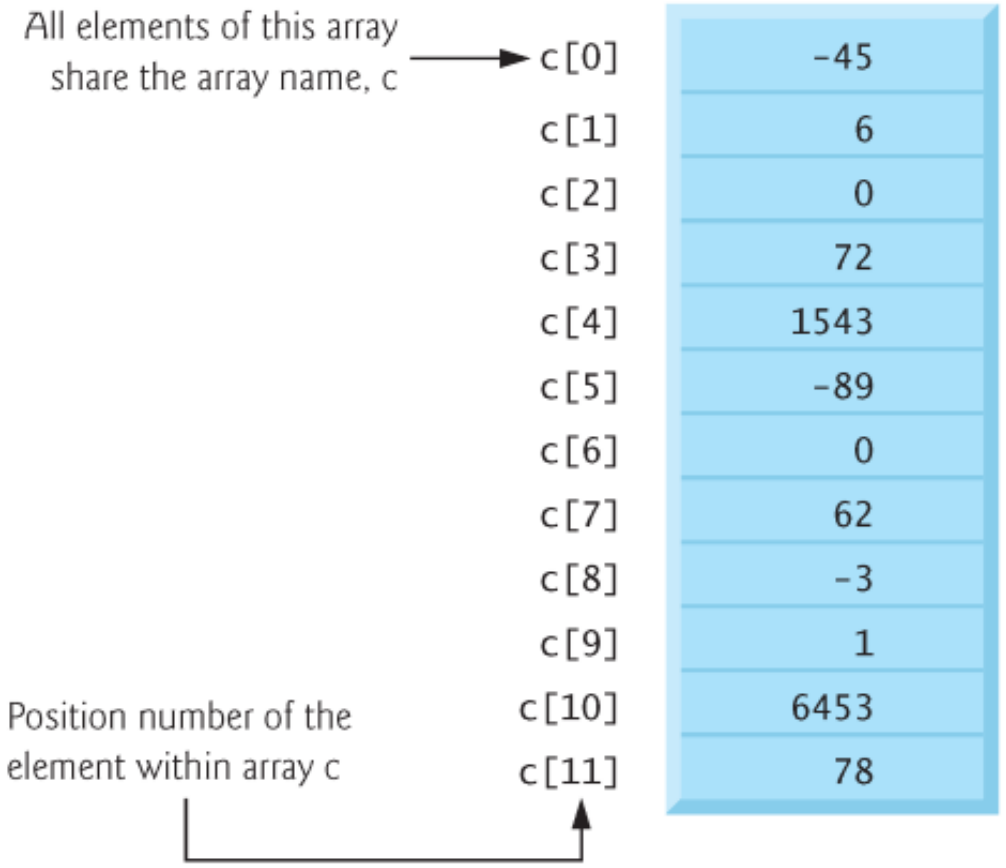
```
type array_name[int_constant_value]={initializer_list};
```

- The size has to be of int type and must be a fixed value (i.e., known at compile time).
  - Static entity – same size throughout program
- You can define an array of any type (eg: int, float, enum student_type, etc.)
- All elements of the array have the same type.

Arrays - Usage

- Array
  - Group of consecutive memory locations
  - Same name and type
- To refer to an element, specify
  - Array name
  - Position number
- Format:

  arrayname[*position number*]
  - First element at position 0
  - n element array named c:
    - c[0], c[1]...c[n – 1]



Array - Access

- Array elements are like normal variables

```
c[ 0 ] =  3;
printf( "%d", c[ 0 ] );
```

- Perform operations in subscript (or index). If `x` equals `3`

```
c[ 5 - 2 ] == c[ 3 ] == c[ x ]
```

Defining Arrays

- When defining arrays, specify
  - Name
  - Type of array
  - Number of elements

    ```
    arrayType arrayName[ numberOfElements ];
    ```
  - Examples:

    ```
    int c[ 10 ];
    float myArray[ 3284 ];
    ```
- Defining multiple arrays of same type
  - Format similar to regular variables
  - Example:

    ```
    int b[ 100 ], x[ 27 ];
    ```

Arrays – Initialization

- The elements of a local array are arbitrary (as all other local variables).
- The elements of a global array are initialized to zero by default (as all other global variables).

- You may initialize an array during definition as follows:

```
int array[5] = {10, 8, 36, 9, 13};
```

- However, you cannot perform such an initialization after the definition, i.e.,

```
int array[5];
array = {10, 8, 36, 9, 13};
```

is syntactically wrong.

Arrays – Initialization Example

- Initializers

```
int n[ 5 ] = { 1, 2, 3, 4, 5 };
```

  - If there are fewer initializers than elements in the array, the remaining elements are initialized to zero.

```
int n[ 5 ] = { 0 }
```

    - All elements 0
  - If there are more initializers than the size of the array, a syntax error is produced.

- If the array size is *omitted* from a definition with an initializer list, the number of elements in the array will be the number of elements in the initializer list.

```
int n[ ] = { 1, 2, 3, 4, 5 };
```

  - 5 initializers, therefore 5 element array

Arrays – Index and bounds

- The index must be of int type.

```
int k[5];
k[k[4]/k[1]]=2;
k[1.5] = 3;      /* Error since 1.5 is not int */
```

- *C has no array bounds checking to prevent the program from referring to an element that does not exist.*
  - Thus, an executing program can "walk off" either end of an array without warning
  - You should ensure that all array references remain within the bounds of the array.

Arrays Have Fixed Size!

- The size of an array *must* be stated at compile time.
- This means you cannot define the size when you run the program. You should fix it while writing the program.
- This is a very serious limitation for arrays. Arrays are not fit for dynamic programming.
  - You should use pointers for this purpose.
- Examples:
  - Initialize_zero.c
  - Initialize_value.c

Example 1: Initialize_zero.c

```
// Initializing the elements of an array to zeros.
#include <stdio.h>

// function main begins program execution
int main(void)
{
   int n[5], i; // n is an array of five integers

   // set elements of array n to 0
   for (i = 0; i < 5; ++i) {
      n[i] = 0; // set element at location i to 0
   }

   printf("%s\t%s\n", "Element", "Value");

   // output contents of array n in tabular format
   for (i = 0; i < 5; ++i) {
      printf("%d\t%d\n", i, n[i]);
   }
}
```

Output:

```
Element Value
0       0
1       0
2       0
3       0
4       0


------------------------------
```

Example 2: Initialize_value.c

```
#include <stdio.h>

// function main begins program execution
int main(void)
{
   // use initializer list to initialize array n
   int x;
   int i, n[5] = {32, 27, 64, 18, 95};

   printf("%s\t%s\n", "Element", "Value");

   // output contents of array in tabular format
   for (i = 0; i < 5; ++i) {
      printf("%d\t%d\n", i, n[i]);
   }
   printf("\nValues:\n");
   printf("%d\t%d\t%d\t%d\t%d\n", *n, *(n+1),*(n+2),*(n+3),*(n+4));

   printf("\nAddresses:\n");
   printf("%p\t%p\t%p\t%p\t%p\n", n, n+1, n+2, n+3, n+4);
   printf("%p\t%p\t%p\t%p\t%p\n", &n[0], &n[1], &n[2], &n[3], &n[4]);
}
```

Output of Initialize_value.c

Output:

```
Element Value
0       32
1       27
2       64
3       18
4       95

Values:
32      27      64      18      95

Addresses:
000000000062FE00        000000000062FE04        000000000062FE08  000000000062FE0C
000000000062FE10
000000000062FE00        000000000062FE04        000000000062FE08  000000000062FE0C
000000000062FE10

--------------------------------
```

Variable array size

- At declaration, the size of the array can be given as a variable or an expression

```
int arsize;
scanf("%d", &arsize);

int array[arsize];
for(int i=0; i<arsize; i++)
     array[i] = i*i;
```

- Works only with compilers compatible with the C99 standard.
  - You should add –std=c99 compiler option

- An array's size cannot be changed once it has been created.

Example 3: Histogram

```
#define SIZE 5

int main(void){
   // use initializer list to initialize array n
   int i, j, n[SIZE] = {19, 3, 15, 7, 11};

   printf("%s\t%s\t%s\n", "Element", "Value", "Histogram");

   for (i = 0; i < SIZE; ++i) {
      printf("%d\t%d\t", i, n[i]) ;

      for (j = 1; j <= n[i]; ++j)
         printf("%c", '*');

      printf("\n");
   }
}
```

Output:

```
Element Value   Histogram
0       19      ********************
1       3       ***
2       15      ***************
3       7       *******
4       11      ***********

--------------------------------
```

Example 4: Roll dice

```
#define SIZE 7
int main(void){
   unsigned int frequency[SIZE] = {0}; // clear counts
   unsigned int roll, face;
   srand(time(NULL)); // seed random number generator

   // roll die 600 times
   for (roll = 1; roll <= 600; ++roll) {
      face = 1 + rand() % 6;
      ++frequency[face];
   }

   printf("%s\t%s\n", "Face", "Frequency");
   for (face = 1; face < SIZE; ++face) {
      printf("%d\t%d\n", face, frequency[face]);
   }
}
```

Output:

```
Face    Frequency
1       94
2       100
3       103
4       103
5       96
6       104

----------------------
```

Arrays as Parameters

- Although you write like a value parameter, an array is always passed by reference (variable parameter).
  - Therefore, when you make a change in an element of an array in the function, the change is visible from the caller.

Passing Arrays to Functions

- Passing arrays
  - To pass an array argument to a function, specify the name of the array without any brackets

```
int myArray[ 24 ];
myFunction( myArray, 24 );
```

    - Array size usually passed to function
  - Arrays passed call-by-reference
  - The name of array is address of first element
  - Function knows where the array is stored
    - Modifies original memory locations
- Passing array elements
  - Passed by call-by-value
  - Pass subscripted name (i.e., `myArray[ 3 ]`) to function

Passing Arrays to Functions – cont.

- Function prototype

```
void modifyArray( int b[], int arraySize );
```

  - Parameter names optional in prototype
    - `int b[]` could be written `int []`
    - `int arraySize` could be simply `int`

- Examples:
  - Initialize_function.c
  - Initialize_function2.c
  - Initialize_function_size.c
  - Modify_array.c

Example 5 – Initialize_function.c

- Fill in an array of integer from input.

```
#include <stdio.h>

void read_array(int ar[10])
{  int i;
   for (i=0; i<10; i++)
      scanf("%d", &ar[i]);
}
int main()
{  int a[10], i;
   read_array(a);
   for (i=0; i<10; i++)
      printf("%d ",a[i]);
   return 0;
}
```

Example 6 – Modify_array.c

```
#define SIZE 5
void modifyArray(int b[], int size);
void modifyElement(int e);

int main(void){
   int a[SIZE] = { 0, 1, 2, 3, 4 };
   int i;
   printf("The values of the original array are:\n");
  for (i = 0; i < SIZE; ++i) {
      printf("%d\t", a[i]);
   }

   modifyArray(a, SIZE);
   printf("\nThe values of the modified array are:\n");
   for (i = 0; i < SIZE; ++i) {
      printf("%d\t", a[i]);
   }

   printf("\n\n\nEffects of passing array element by value:\n\nThe value of a[3] is %d\n", a[3]);
   modifyElement(a[3]);
   printf("The value of a[3] is %d\n", a[3]);
}
```

```
void modifyArray(int b[], int size){
    int j;
   // multiply each array element by 2
   for (j = 0; j < size; ++j) {
      b[j] *= 2;
   }
}

void modifyElement(int e){
   // multiply parameter by 2
   printf("Value in modifyElement is %d\n", e *= 2);
}
```

Example 6 – Modify_array.c (cont.)

Output:

```
The values of the original array are:
0       1       2       3       4

The values of the modified array are:
0       2       4       6       8


Effects of passing array element by value:

The value of a[3] is 6
Value in modifyElement is 12
The value of a[3] is 6

-----------------------------
```

Checking equality of arrays

- To check the equality of two arrays, we need to compare elements pairwise in a loop.

- A comparison such as

```
a==b
```

does not work when a and b are arrays.

Checking equality of arrays (cont.)

- Write a function that returns 1 if two arrays are equal, 0 otherwise

```
int arraycomp(int a[], int b[], int size){
      for(int i=0; i<size; i++)
            if(a[i] != b[i])
                  return 0;
      return 1;
}
```

- Now, given arrays a and b, the function call arraycomp(a, b, n) can be used inside if statement.

Checking equality of arrays (cont.)

```
#include <stdio.h>
int arraycomp(int a[], int b[], int size){
      for(int i=0; i<size; i++)
            if(a[i] != b[i])
                  return 0;
      return 1;
}
int main(){
      int a1[5] = {3,7,2,8,5};
      int a2[5] = {3,7,2,8,5};
      int a3[5] = {3,7,2,5,8};
      printf("a1 == a2 is %s\n", arraycomp(a1, a2, 5) ? "True":"False");
      printf("a1 == a3 is %s\n", arraycomp(a1, a3, 5) ? "True":"False");
      printf("a2 == a3 is %s\n", arraycomp(a2, a3, 5) ? "True":"False");
}
```

Copying arrays

- Suppose we have two arrays, declared to be of the same type and of the same length.

```
int a[10] = {1,2,3,4,5,6,7,8,9,10};
int b[10];
```

- We want to copy the contents of a to b.

- A plain assignment

```
b = a;
```

does not work.

- We need to assign element wise.

Copying arrays

- Write a function that copies the contents of array ***src*** to array ***dest***

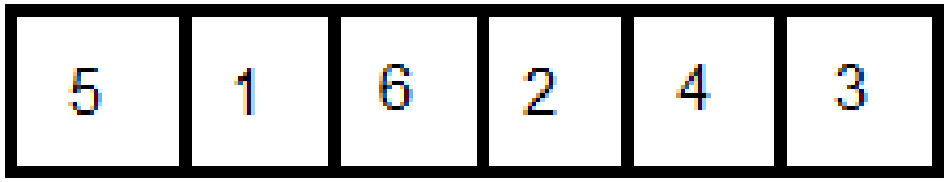
```
#include <stdio.h>
void arraycopy(int src[], int dest[], int size){
      for(int i=0; i<size; i++)
            dest[i] = src[i];
}

int main(){
      int a1[5] = {3,7,2,8,5};
      int a2[5];
      arraycopy(a1, a2, 5);
      for(int i=0; i<5; i++)
            printf("%d ", a2[i]);
}
```

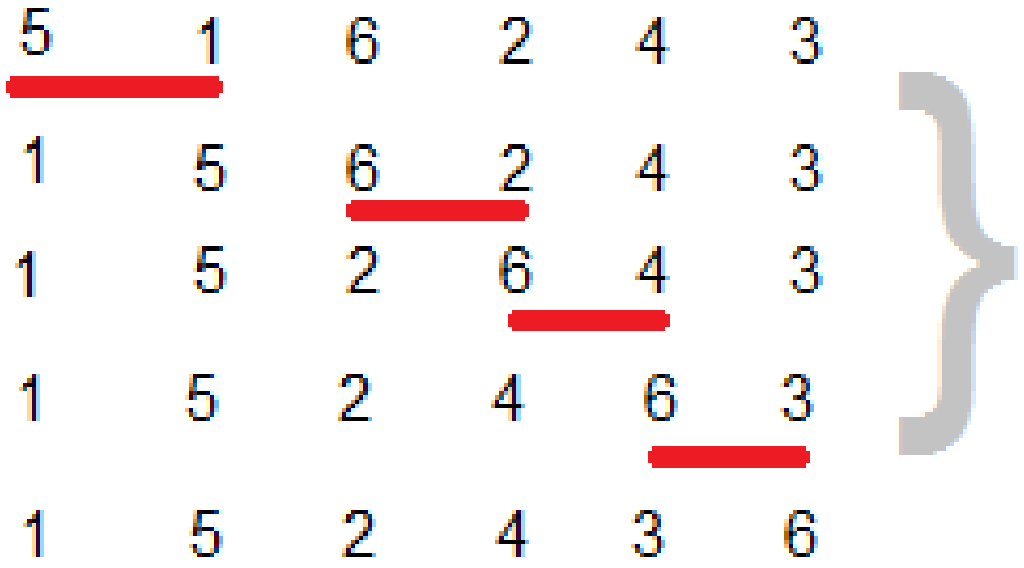
Sorting Arrays – Bubble Sort

- Sorting data
  - Important computing application
  - Virtually every organization must sort some data
- Bubble sort (sinking sort)
  - Several passes through the array
  - Successive pairs of elements are compared
    - If increasing order (or identical), no change
    - If decreasing order, elements exchanged
  - Repeat
- Example:
  - original: 3 4 2 6 7
  - pass 1: 3 2 4 6 7
  - pass 2: 2 3 4 6 7
  - Small elements "bubble" to the top

Bubble Sort



Lets take this Array.



Here we can see the Array after the first iteration.

Similarly, after other consecutive iterations, this array will get sorted.

Example – Bubble Sort

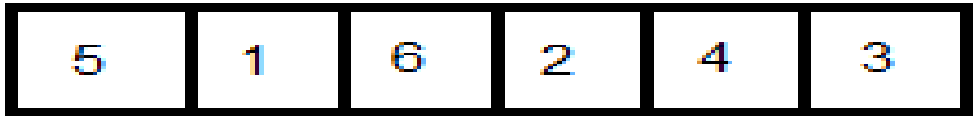
```
void read_array(int ar[], int size)
{  int i;
   for (i=0; i<size; i++)
      scanf("%d", &ar[i]);
}
void print_array(int ar[], int size)
{  int i;
   printf("Sorted Array:\n");
   for (i=0; i<size; i++)
      printf("%d\t", ar[i]);
   printf("\n");
}
void swap(int *a, int *b)
{  int temp;
   temp = *a;
   *a = *b;
   *b = temp;
}
```

Example – Bubble Sort (cont'd)

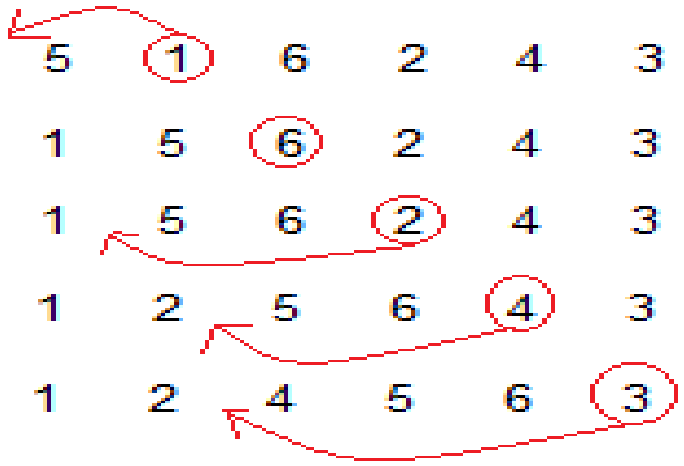
```
void bubble_sort(int ar[], int size)
{  int i, j;
   for (i = 0; i < size-1; i++)
      for (j = 0; j < size-1-i; j++)
         if (ar[j] > ar[j+1])
            swap(&ar[j],&ar[j+1]);
}

int main()
{  int ar[10];
   read_array(ar,10);
   bubble_sort(ar,10);
   print_array(ar,10);
   return 0;
}
```

Insertion Sort



Lets take this Array.



(Always we start with the second element as key.)

As we can see here, in insertion sort, we pick up a key, and compares it with elemnts ahead of it, and puts the key in the right place

5 has nothing before it.

1 is compared to 5 and is inserted before 5.

6 is greater than 5 and 1.

2 is smaller than 6 and 5, but greater than 1, so its is inserted after 1.

And this goes on...

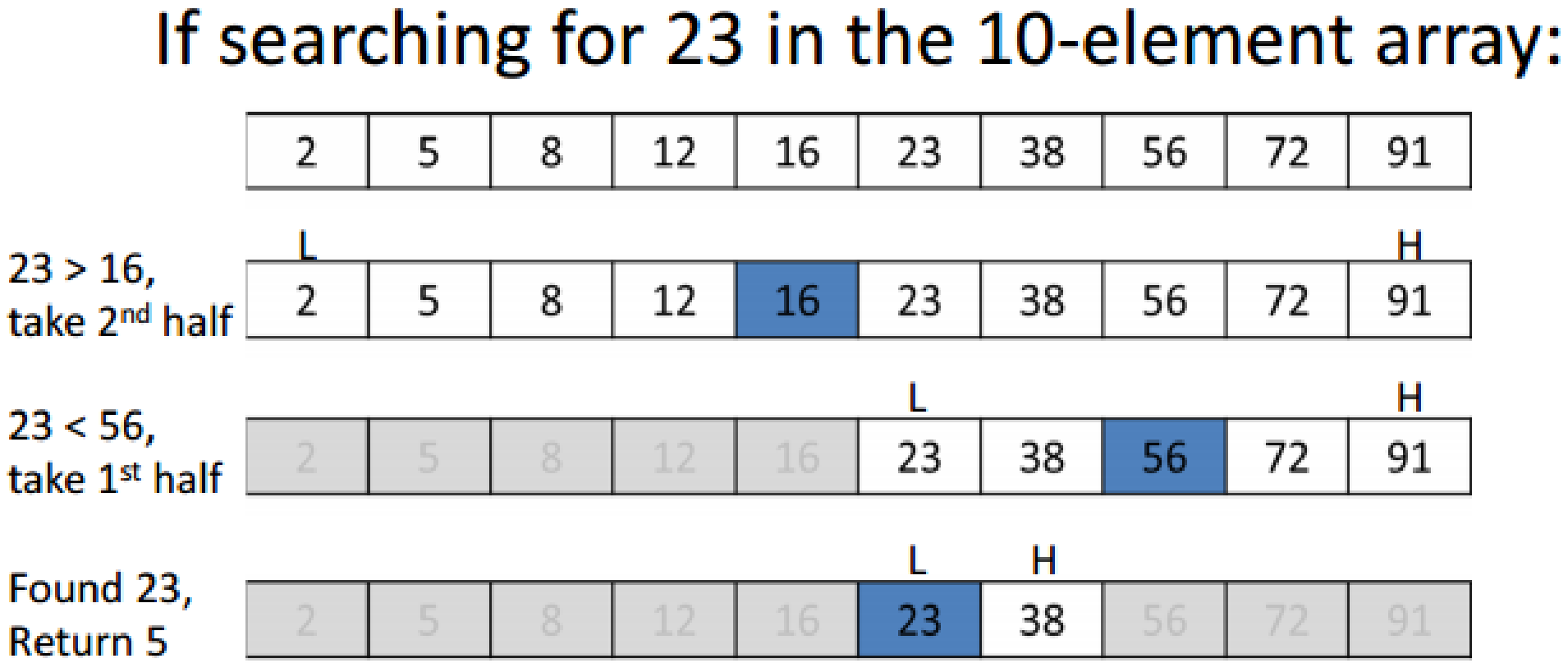
Example – Insertion Sort

```
void insertion_sort(int ar[], int size)
{
   int value, i, j;
   for (i=1; i<size; i++)
   {
      value = ar[i];
      j = i-1;
      while ((j>=0) && (ar[j]>value))
      {
         ar[j+1] = ar[j];
         j--;
      }
      ar[j+1] = value;
   }
}
```

Searching Arrays

- Search an array for a *key value*
- Linear search
  - Simple
  - Compare each element of array with key value
  - Useful for small and unsorted arrays
- Binary search
  - For sorted arrays
  - Compares `middle` element with `key`
    - If equal, match found
    - If `key < middle`, looks in first half of array
    - If `key > middle`, looks in last half
    - Repeat

Binary Search



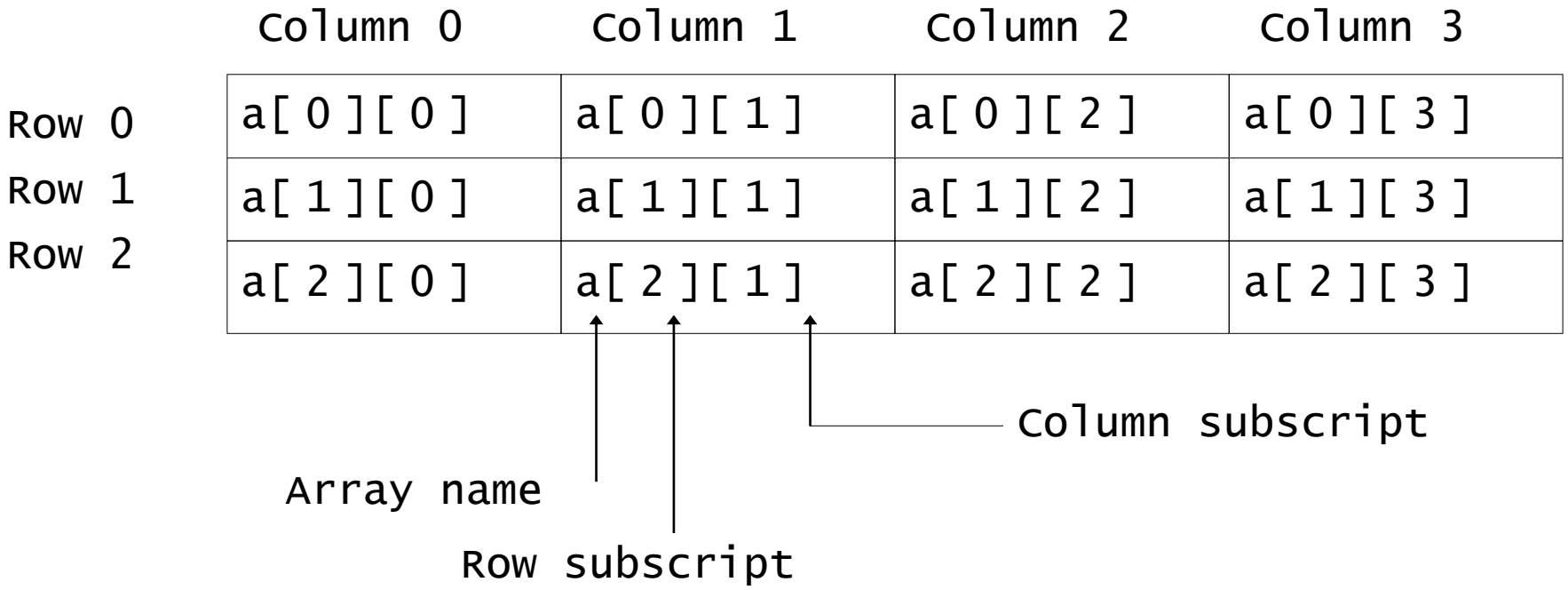
Example – Binary Search

```
int binary_search(int A[], int number, int N)
{  int low = 0, high = N - 1, mid;

   while (low <= high)
   {  mid = (low + high) / 2;
      if (A[mid] == number)
         return mid;
      if (A[mid] < number)
         low = mid + 1;
      else
         high = mid - 1;
   }
   return -1;
}
```

Multidimensional Arrays

- Multiple subscripted arrays
  - Tables with rows and columns (`m` by `n` array)
  - Like matrices: specify row, then column



Multidimensional Arrays – Initialization & Accessing Elements

- Initialization
  - `int b[ 2 ][ 2 ] = { { 1, 2 }, { 3, 4 } };`
  - Initializers grouped by row in braces
  - If not enough, unspecified elements set to zero

    `int b[ 2 ][ 2 ] = { { 1 }, { 3, 4 } };`

1	2
3	4

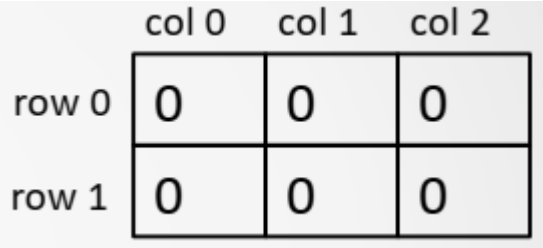
1	0
3	4

- Referencing elements
  - Specify row, then column

    `printf( "%d", b[ 0 ][ 1 ] );`

Setting array elements

- As with regular (1-D) arrays, multidimensional arrays are frequently assigned and read with loops.

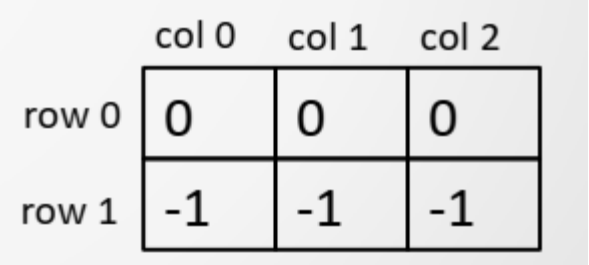


- Declare and initialize to zero

```
int a[2][3] = {0};
```

- Set all elements in the second row (index 1) to minus one.

```
for(int column=0; column < 3; column++)
        a[1][column] = -1;
```



- Set all elements in the first column (index 0) to one

```
for(int row=0; row < 2; row++)
        a[row][0] = 1;
```