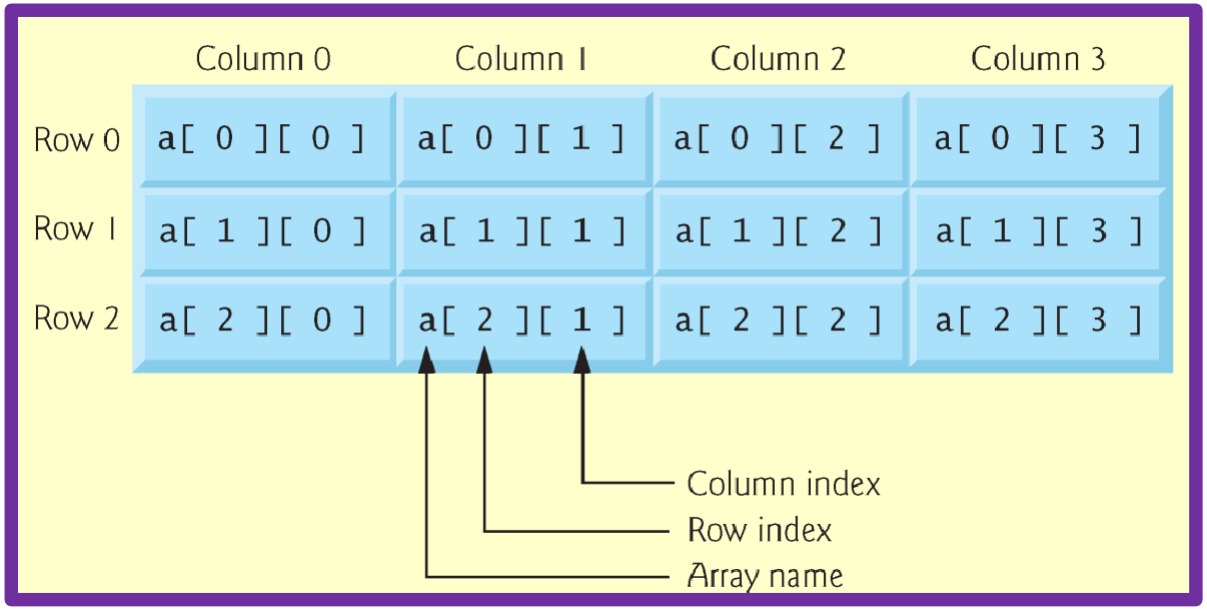
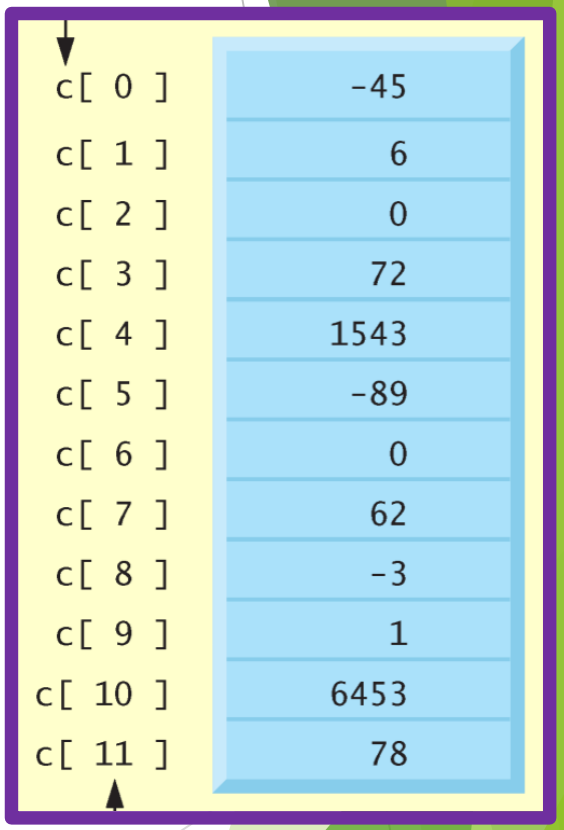
Arrays

Arrays

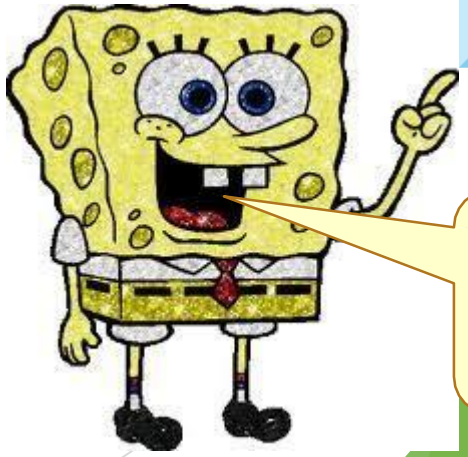




Introduction to Arrays

- A collection of variable data
  - Same name
  - Same type
  - Contiguous block of memory
  - Can manipulate or use
    - Individual variables or
    - ‘List’ as one entity

-45
6
0
72
1543
-89
0
62
-3
1
6453
78



Celsius temperatures: I’ll name it c. Type is int.

Introduction to Arrays

- Used for lists of like items
  - Scores, speeds, weights, etc.
  - Avoids declaring multiple simple variables
- Used when we need to keep lots of values in memory
  - Sorting
  - Determining the number of scores above/below the mean
  - Printing values in the reverse order of reading
  - Etc.

Declaring Arrays

- General Format for declaring arrays

```
<data type>  <variable> [<size>];
```

- Declaration
  - Declaring the array → allocates memory
  - Static entity - same size throughout program
- Examples

```
int c[12];
int scores[300];
float weight[3284];
char alphabet[26];
```

Type is int.
Name is c.

Defined Constant as Array Size

- Use defined/named constant for array size
  - Improves readability
  - Improves versatility
  - Improves maintainability
- Examples:

```
const int NUMBER_OF_STUDENTS = 50;
// ..
int scores[NUMBER_OF_STUDENTS];
```

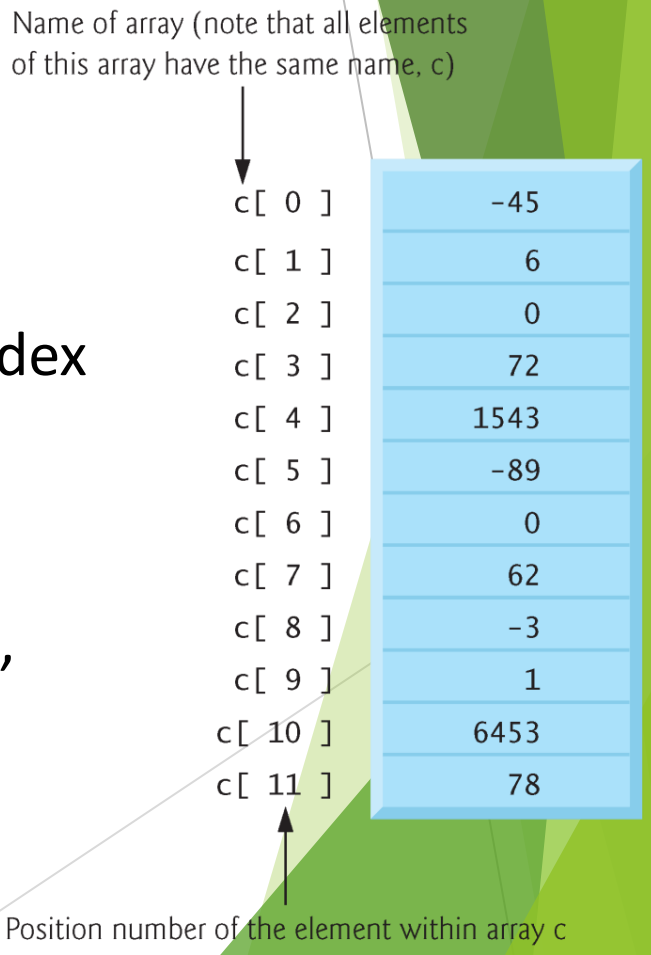
```
#define NUMBER_OF_STUDENTS 50
// ..
int scores[NUMBER_OF_STUDENTS];
```

Powerful Storage Mechanism

- Can perform subtasks like:
  - "Do this to i-th indexed variable" where i is computed by program
  - "Fill elements of array scores from user input"
  - "Display all elements of array scores“
  - “Sort array scores in order”
  - “Determine the sum or average score”
  - "Find highest value in array scores"
  - "Find lowest value in array scores"

Accessing Array Elements

- Individual parts called many things:
  - Elements of the array
  - Indexed or subscripted variables
- To refer to an element:
  - Array name and subscript or index
  - Format: **arrayname[subscript]**
- Zero based
  - `c[0]` refers to c_0, c sub zero, the **first** element of array **c**



Accessing Array Elements

- Example

```
printf("%d\n", c[5]);
```

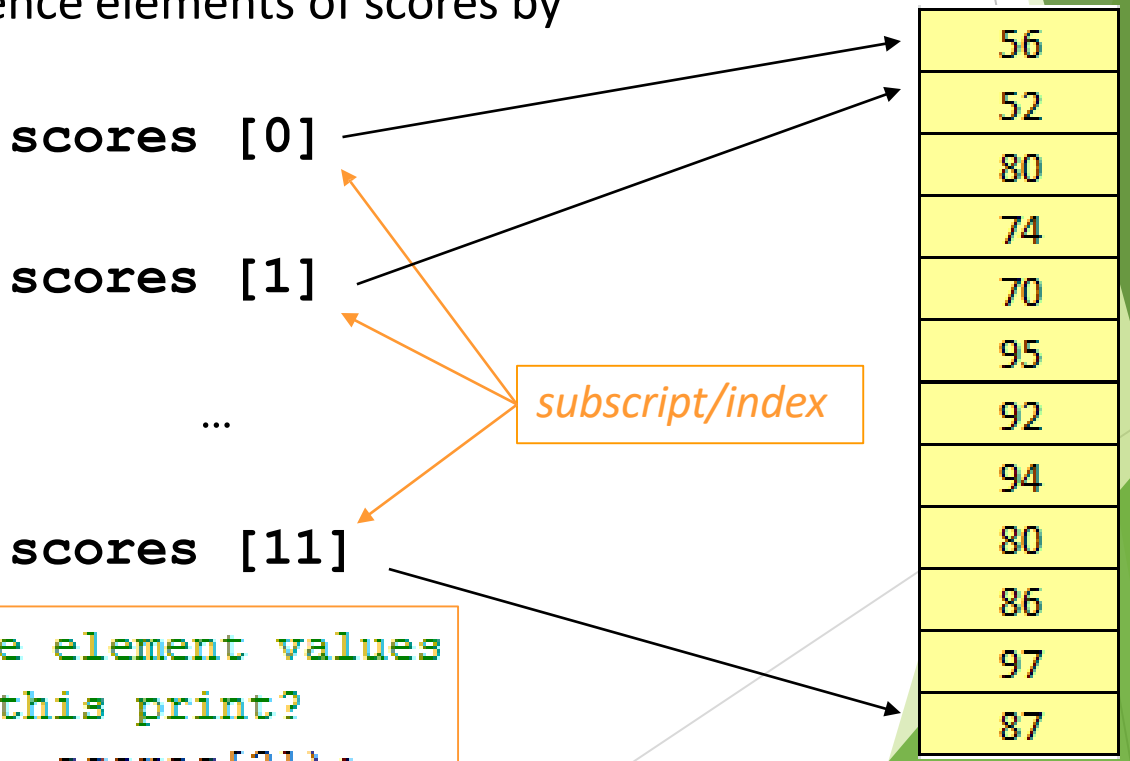
- Note two uses of brackets:
  - In declaration, specifies SIZE of array
  - Anywhere else, specifies a subscript/index

Accessing Array Elements

- Example
  - Given the declaration

```
int scores[12];
```

    - We reference elements of scores by




```
// Given these element values
// What does this print?
printf("%d\n", scores[3]);
```

Accessing Array Elements

- Size, subscript need not be literal constant
  - Can be named constant or expression

```
int scores[MAX_SCORES]; // MAX_SCORES is a constant
scores[n+1] = 99;       // If n is 2, same as scores[3]
```

Major Array Pitfall

- Array indexes go from 0 through size-1!
- C will 'let' you go out of the array's bounds
  - Unpredictable results – may get segmentation fault
  - Compiler will not detect these errors!
- Up to programmer to 'stay in bounds'

```
printf("%d\n",scores[-8]);
scores[250] = 88;
```

?

?

56
52
80
74
70
95
92
94
80
86
97
87

for-loops with Arrays

- Natural counting loop
  - Naturally works well 'counting thru' elements of an array
- General form for forward direction
  - for (subscript = 0; subscript < size; subscript++)
- General form for reverse direction
  - for (subscript = size-1; subscript >= 0; subscript--)

for-loops with Arrays Examples

```
int scoreSub;
// Print forward
for (scoreSub = 0; scoreSub < 12; scoreSub++)
   printf("Score %d is %d\n", scoreSub+1,
              scores[scoreSub]);
// Print backward, in reverse
for (scoreSub = 11; scoreSub >= 0; scoreSub--)
   printf("Score %d is %d\n", scoreSub+1,
              scores[scoreSub]);
```

56
52
80
74
70
95
92
94
80
86
97
87

```
Score 1 is 56
Score 2 is 52
Score 3 is 80
Score 4 is 74
...
Score 12 is 87
```

```
Score 12 is 87
Score 11 is 97
Score 10 is 86
Score 9 is 80
...
Score 1 is 56
```

Uses of Defined Constant

- Use everywhere size of array is needed
  - In for-loop for traversal:

```
int score;
for (score=0; score<NUMBER_OF_STUDENTS; score++)
   printf("%d\n", scores[score]);
```

  - In calculations involving size:

```
lastIndex = NUMBER_OF_STUDENTS - 1;
lastScore = scores[NUMBER_OF_STUDENTS - 1];
```

  - When passing array a function:

```
total = sum_scores(scores, NUMBER_OF_STUDENTS);
```

Array as Function Parameter

- Include type and brackets []
  - Size inside brackets is optional and is ignored
- Passes pointer/reference to array
  - Function can modify array elements
- Common to also pass size
- Example:

```
void print_scores(int values[], int num_values) {
    // Call: print_scores(scores, scoreCount)
    int valueNdx;
    for (valueNdx=0; valueNdx<num_values; valueNdx++)
        printf("%d\n", values[valueNdx]);
}
```

Initializing Arrays

- Arrays can be initialized at declaration

```
int scores[3] = {76, 98, 83};
```

  - Size cannot be variable or named constant
  - Equivalent to

```
int scores[3];
scores[0] = 76;
scores[1] = 98;
scores[2] = 83;
```

Auto-Initializing Arrays

- If fewer values than size supplied:
  - Fills from beginning
  - Fills 'rest' with zero of array base type
    - Declaration

```
int scores[5] = {76, 98, 83}
```

    - Performs initialization

```
scores[0] = 76;
scores[1] = 98;
scores[2] = 83;
scores[3] = 0;
scores[4] = 0;
```

Auto-Initializing Arrays

- If array size is left out
  - Declares array with size required based on number of initialization values
  - Example:

```
int scores[] = {76, 98, 83}
```

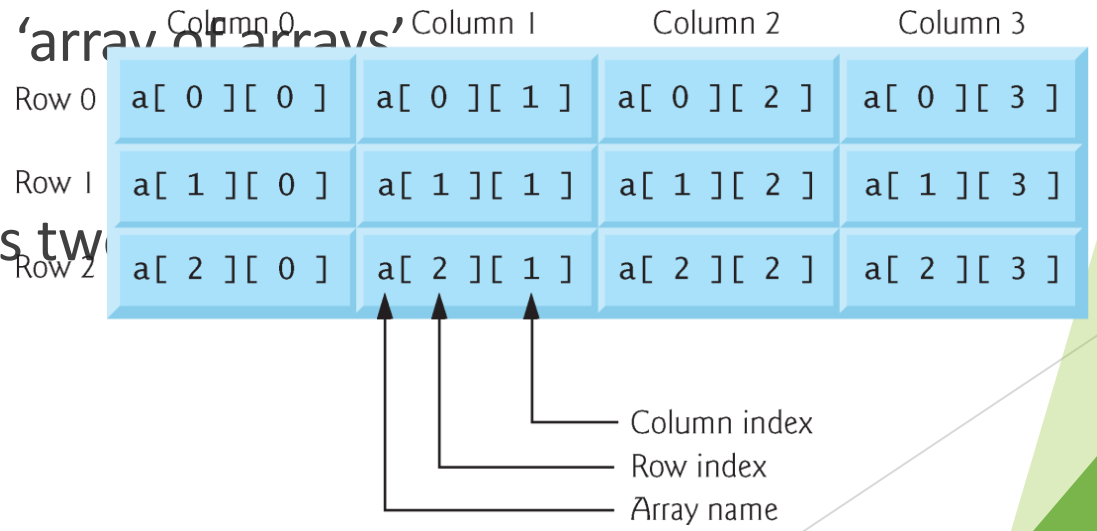
    - Allocates array scores with size of 3

Multidimensional Arrays

- Arrays with more than one dimension
  - Declaration: Additional sizes each enclosed in brackets

```
int a[3][4];
```

- Two dimensions
  - Table or 'array of arrays'
  - Requires two



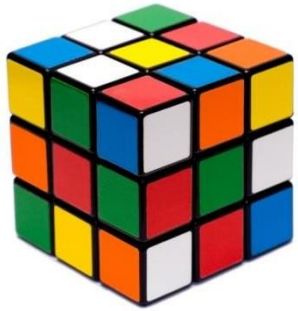
Initializing Multidimensional

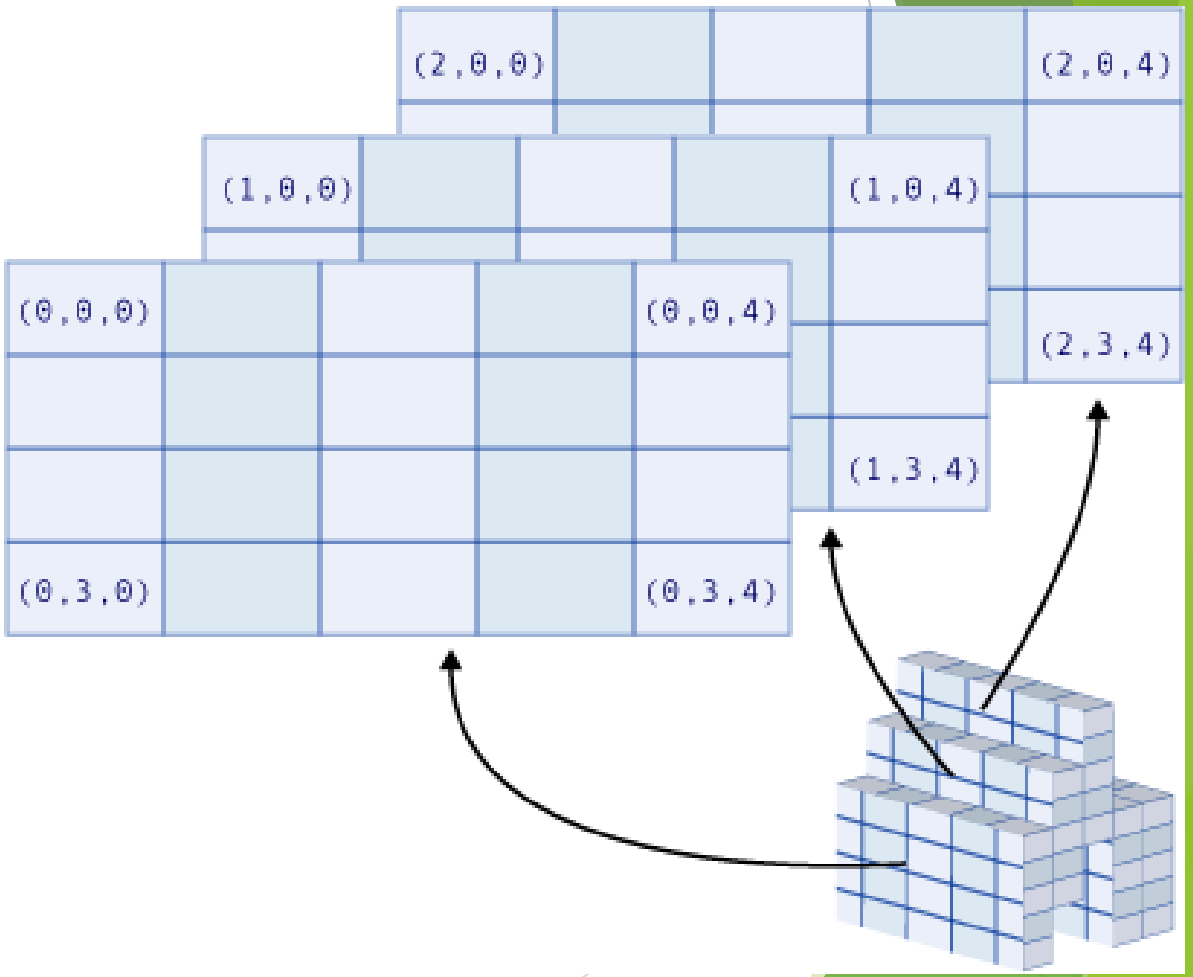
- Nested lists
  - Unspecified values set to zero
- 2D Example:

```
int nums[4][5] = { {10,  6, -7, 13, 28},
                   {10,  5, 44,  8},
                   {33, 20,  1,  0, 14},
                   { 2, 66, 25, 37,  1}
                 }
```

Three-dimensional Visualization

```
int cube[3][3][3];
```





Multidimensional Array Parameters

- Must specify size after first dimension

```
void scalar_multiply(int rows, int cols,
                             int a[][cols], int scalar) {
   // multiplies each element in array by scalar
   int row, col;
   for (row=0; row<rows; row++)
      for (col=0; col<cols; col++)
         a[row][col] *= scalar;
}
```