**Arrays**

An array is an aggregate of multiple elements of similar type. They’re use to store or allocate a certain amount of memory for a certain size or amount of variables.

**Array Declaration**:

‘type’’name’[‘size of the array’];

**Array Initialization Example 1 (All At Once):**

int Example[3] = {1, 2, 3};  
 // Assigns index’s starting from 0 to 2

**Array Initialization Example 2 (Independently):**

int Example[3] //Declaration

int Example[0] = 1; //Index 0 = 1

int Example[1] = 2; //Index 1 = 2

int Example[2] = 3; //Index 2 = 3

**Array Initialization Example 3 (Looping):**

const int Capacity = 3;  
 // Use a Constant variable to prevent changing of array size

int variable = 1; //Set starting variable here

int Example[Capacity] // Declaration

for(int index = 0; index < Capacity; index++){

// Starting loop from index of 0 to index of 2

Example[index] = {variable++};}

**Array Output Example 1 (One by One):**

cout << Example[0]; // Outputs: 1

cout << Example[1] << Example[2]; // Outputs: 23

**Array Output Example 2 (All - Looping):**

for(int index = 0; index < Capacity; index++){

cout << Example[index] << " ";} // Outputs: 1 2 3

**Multi-Dimensional Arrays**

Multi-Dimensional Arrays are arrays of arrays. There’s an unlimited amount of dimensions multi-dimensional arrays can become, but the simplest are the two-dimension arrays.

**Array Declaration**:

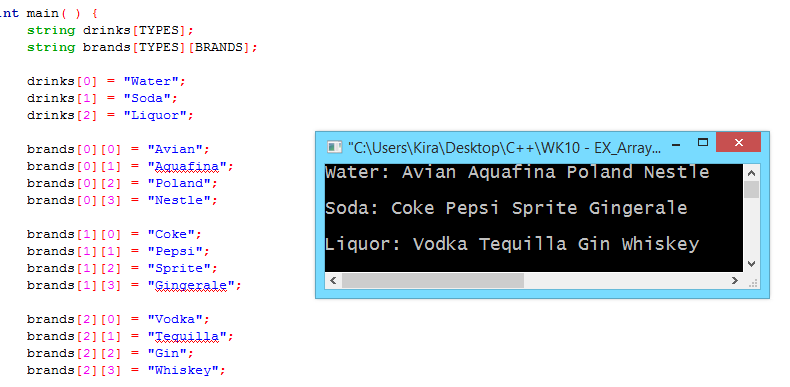
‘type’’name’[‘Array Group’][‘Index’];

**Array Initialization Example 1 (All At Once)::**

int Example[2][2] = {1,2,3,4};  
 // There’s two arrays, both with an index of 2.

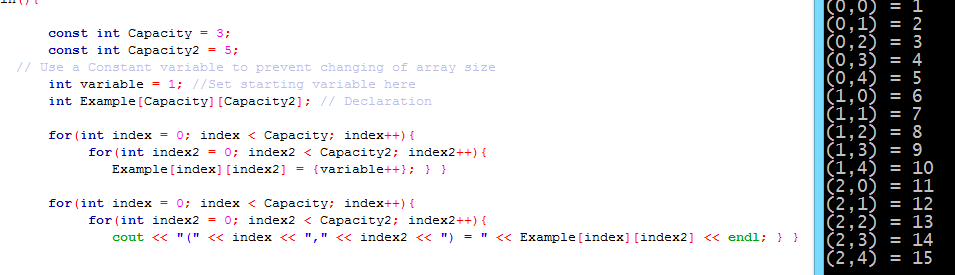
// There’s Array 0 and Array 1, Array one has index of 0 and 1

**Array Initialization Example 2 (Independently): :**



//Above shows a single array called Drinks, and the 2-dimensional called brands

**Array Initialization/Output Looping Example:**



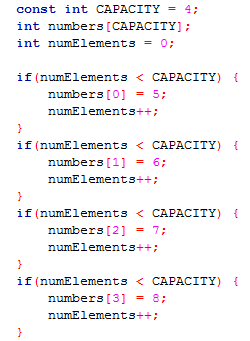
**Partially-Filled Arrays**

Partially filled arrays is exactly what it sounds like. In order to output correctly though the user needs to create another variable indicating how many indexes are filled.

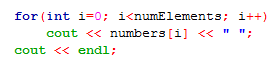
**Example 1 (All At Once)**:



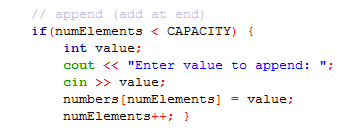
**Example 2 (Independently)**:  
// If the index is less than capacity, then index of number = value and increase amount of index by one.



**Looping Output**

// Output each array starting from 0 until index is less than numElements (amount of filled elements) 

**Append Element**:  
 numElements is the amount of used indexes (starting from 0 to index-1) thus accrediting a value to the last index and afterwards increasing the index appends the array)



**Delete Last Element**:

Like hard drives, Arrays cannot be fully deleted unless replaced by other data. To ‘delete’ the last element without replacing the data is to decrease the variable indicating the amount of used indexes.   
   
 numElements--;

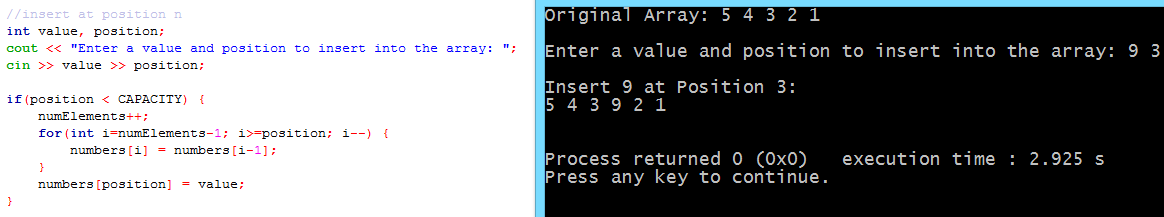
**Delete First Element**:

To delete the first element, you have to create a loop that shifts each of the afterward elements over, up one (*0 = 1, 1 = 2, 2 = 3*), and reduce the amount of indexes filled by one.

// Loop (index = position of element to be removed; index < amount of indexes filled, index + 1) {  
 array[index] = array[index + 1];  
 amount of indexes filled – 1;

  
  
//Result: array[5] = 20;  
 array[6] = 30;  
 array[5] = array[6]  
 array [5] = 30

**Insert Element At Position**:



Updated 9/12/2017