**Arrays**

An array is a contiguous space in memory to store values. An array is an ordered sequence of values. The order is indicated by the position or index. The first position or index starts at 0. The next index 1 and so on. The last index is the size of the array - 1. Here is an example of an array of 10 unsorted integers.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| value | 12 | 49 | -2 | 26 | 5 | 17 | -6 | 84 | 72 | 3 |

### Array declaration and creation

When creating an array, you specify the number of elements in the array as follows:

***variable* = new *type*[*length*];**

For example, to create an array of 10 integers:

numbers = new int[10];

### Storing Values and Accessing Elements

The syntax for storing a value in an array element is:

***variable*[*index*] = *expression*;**

For example:

numbers[0] = 27;

numbers[3] = -6;

would change the numbers array to:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| value | 27 | 0 | 0 | -6 | 0 | 0 | 0 | 0 | 0 | 0 |

The syntax for accessing an array element is:

***variable*[*index*]**

Where the index can be any expression that results in an int. For example:

if (numbers[3] > 0) {

Print (numbers[3] + " is positive");

} else {

Print (numbers[3] + " is not positive");

}

When you declare an array, each element of the array is set to a default initial value.  
For integers, the default value is 0.

**Arrays and Loops**

We can use an integer variable as the index of an array.  If we use a for loop to count from 0 to the highest index, then we can process each element of an array.  For example, the following code would sum the elements in the numbers array.

int sum = 0;

for I = 0 to array length - 1

sum = sum + numbers[i]

Next

### Find Maximum Value in the Array

To find the maximum value, you initialize a placeholder called *max* with the value of the first element in the array. Then you go through the array element by element. If any element is greater than *max* you replace *max* with that element. Here is the pseudocode:

Max = array[0]

FOR i = 1 to array length - 1

IF array[i] > Max THEN

Max = array[i]

ENDIF

Next

PRINT Max

**Sequential Search for an Element in the Array**

Let the element that we are searching for be MAX. We need to know if that element occurs in the array. It will return the position of the occurrence of that element. This is the pseudo code:

FOR i = 0 to array length - 1

IF MAX = array[i] THEN

PRINT i

ENDIF

Next

Another variation of this problem is to return the number of occurrences of X in the array. Here is a modification of the above code:

SET Count to 0

FOR i = 0 to array length - 1

IF X = array[i] THEN

COUNT = COUNT +1

ENDIF

NEXT

PRINT Count

**Selection Sort**

This is one of the easiest sorting algorithms to understand and write but not the most efficient one. In Selection Sort we start at the first element of the array and go through the array and find the minimum element. We swap the minimum element with the element at the first place. We start at the second position in the array and go through the array and find the minimum element in the remaining portion of the array. We swap that minimum element with the element with the element at the second position. We start at the third position and repeat the procedure until we reach the end of the array.

FOR i = 0 to array length - 2

Min = array[i]

MinIndex = i

FOR j = i + 1 to array length - 1

IF array[j] < Min THEN

Min = array[j]

MinIndex = j

ENDIF

NEXT

array[MinIndex] = array[i]

array[i] = Min

NEXT

Selection Sort is not efficient. It does the same amount searches if the values in the array are in random order, partially sorted or completely sorted.

**The Temperature Program**

Consider the interaction at the beginning of these notes.  Here is pseudocode that follows the sequence of interactions, using an array to manage the values that the user enters.  Note that we can't count how many elements are above the average until we have computed the average, and we can't compute the average until we have input all the elements.

1. Input the number of days from the user.
2. Declare and create an int array with the number of days as its length.
3. For each index in the array:
   1. Input the temperature from the user.
   2. Store the temperature in the array at that index.
4. Initialize a sum to zero.
5. For each index in the array:
   1. Add the value at that index to the sum.
6. Calculate and print the average.
7. Initialize a counter to zero.
8. For each index in the array:
   1. If the value at that index is greater than the average:
      1. Increment the counter.
9. Print the counter.

We could have combined the first two loops into one loop, but it is cleaner to do them separately.

### Additional Array Features

If you know in advance what the values are going to be in an array, you can specify those values when you declare the array, for example:

int[ ] daysInMonth = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

String[ ] weekDayNames = {"Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"};