PASSING ARRAYS TO FUNCTIONS

To pass an array argument to a function, specify the name of the array without any brackets.
ex: If an array **hourlyTemperatures** has been declared as the function, the call passes array hourlyTemperatures and it's size to function **modifyArray**.

Important Points

- When passing an array to a function, normally the array size is passed as well, so the function can process the specific number of elements in the array. Otherwise, we would need to build this knowledge into the called function itself or, worse yet, place the array size in a global variable.

- C++ passes arrays to functions by reference i.e. the called functions can modify the element values in the callers' original arrays.

- The value of the name of the array is the address in the computer's memory of the first element of the array. Since the starting dress of the array is passed, the called function knows precisely where the array is stored in the memory. Therefore, when the called function modifies array elements in its function body, it is modifying the actual elements of the array in their original memory locations.

- Although the entire arrays are passed by reference, individual array elements are passed by value exactly as simple variables are.

- To pass an element of an array through a function call, the function's parameter list must specify that the function expects to receive an array.

- ex: the function header for function modifyArray might be written as-
  void modifyArray(int b[ ], int arraySize)

- The statement indicates that modifyArray expects to receive the address of an array of integers in parameter b and the number of array elements in parameter arraySize. The array's size is not required in the array brackets. If it is included, the compiler ignores it, thus, arrays of any size can be passed to the function.

- C++ passes arrays to the functions by reference. When the called function uses the array name b, it refers to the actual array in the caller (i.e., arrayhourlyTemperatures discussed at the beginning of the section).

EXAMPLE

```
void modifyArray( int [], int ); // appears strange; array and size
void modifyElement( int ); // receive array element value

void setup () {
   Serial.begin (9600);
   const int arraySize = 5; // size of array a
   int a[ arraySize ] = { 0, 1, 2, 3, 4 }; // initialize array a
   Serial.print ( "Effects of passing entire array by reference:" ) ;
   // output original array elements
   for ( int i = 0; i < arraySize ; ++i )
   Serial.print ( a[ i ] ) ;
   Serial.print ("\r" ) ;
   Serial.print ("The values of the modified array are:\n" );
   // output modified array elements
   for ( int j = 0; j < arraySize; ++j )
   Serial.print ( a[j ] ) ;
   Serial.print ("\r" ) ;
   Serial.print ("\r\rEffects of passing array element by value:" );
   Serial.print ( "\ra[3] before modifyElement: " );
   Serial.print ( a[ 3 ] );
   Serial.print ( "\ra[3] after modifyElement: " );
   Serial.print ( a[ 3 ] );
}
void loop () {

}

// in function modifyArray, "b" points to the original array "a" in memory

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

// end function modifyArray
// in function modifyElement, "e" is a local copy of
// array element a[ 3 ] passed from main

void modifyElement( int e ) {
   // multiply parameter by 2
   Serial.print ( "Value of element in modifyElement: " );
   Serial.print ( ( e *= 2 ) );
}

// end function modifyElement
```

RESULT

Effects of passing entire array by reference:01234
The values of the modified array are:01234

Effects of passing array element by value:
a[3] before modifyElement: 3
a[3] after modifyElement: 3
$ is not a hexadecimal digit
f is a hexadecimal digit

This program demonstrates the difference between passing an entire array and passing an array element.