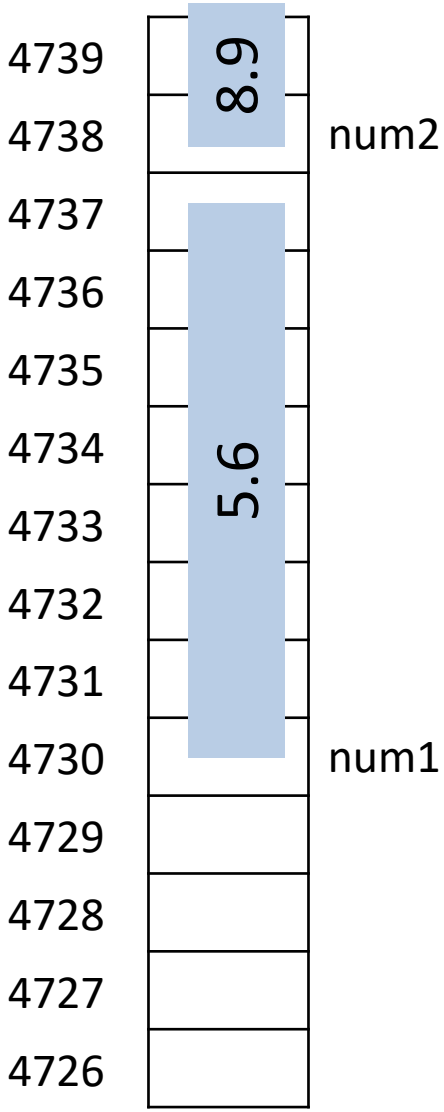
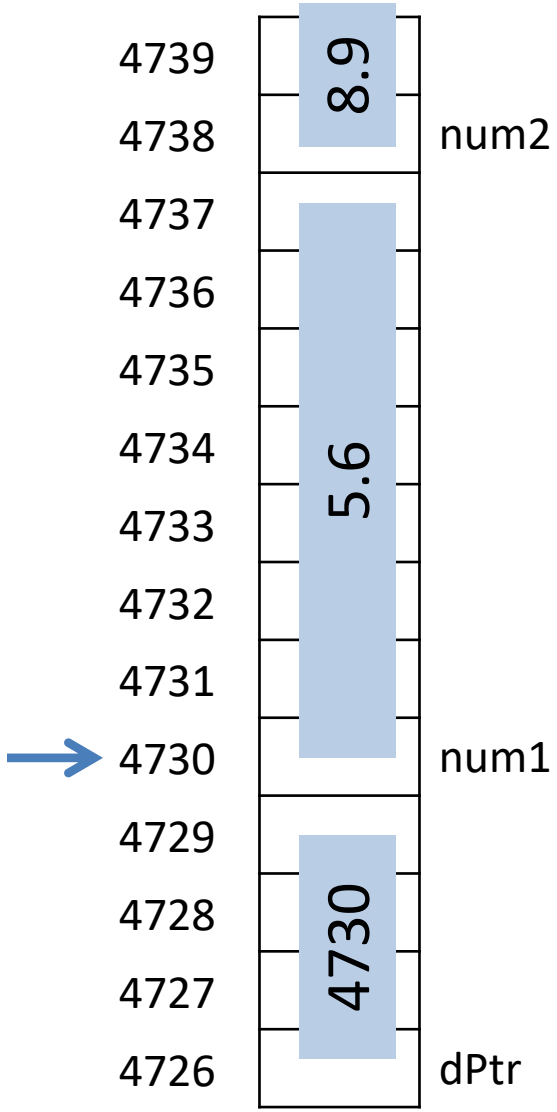
How traditional double variables are stored in RAM, as well as what a POINTER is



double num1 = 5.6, num2 = 8.9;

When “num1” is declared, an 8-byte location in memory is established to store the value of 5.6, and “num1” is an ID that always references that fixed storage location

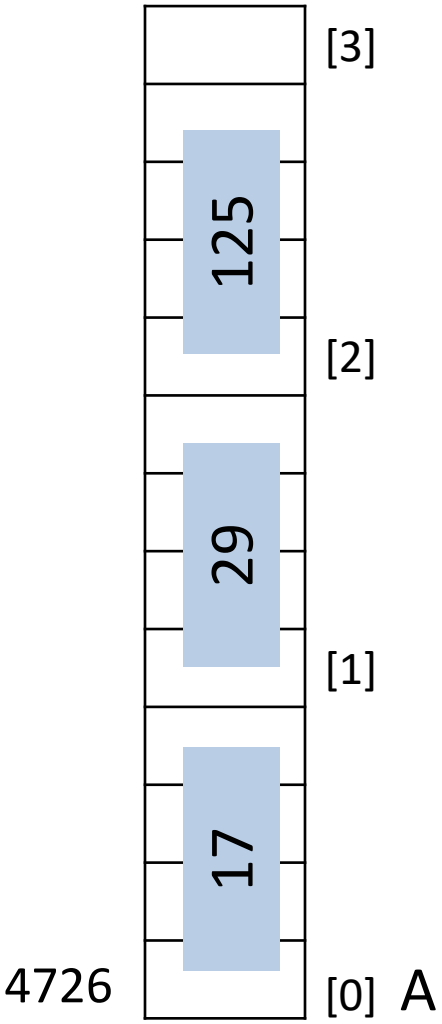
How traditional double variables are stored in RAM, as well as what a POINTER is




```
double  num1 = 5.6, num2 = 8.9;
double  * dPtr;
dPtr = &num1;
num1 == *dPtr  // This now true!
```

When “dPtr” is declared, a storage location is established to hold a pointer to a double.
When dPtr is assigned the value of the ADDRESS of num1, the value of 4730 is stored at dPtr.
When *dPtr is used, it refers to what is POINTED TO by dPtr

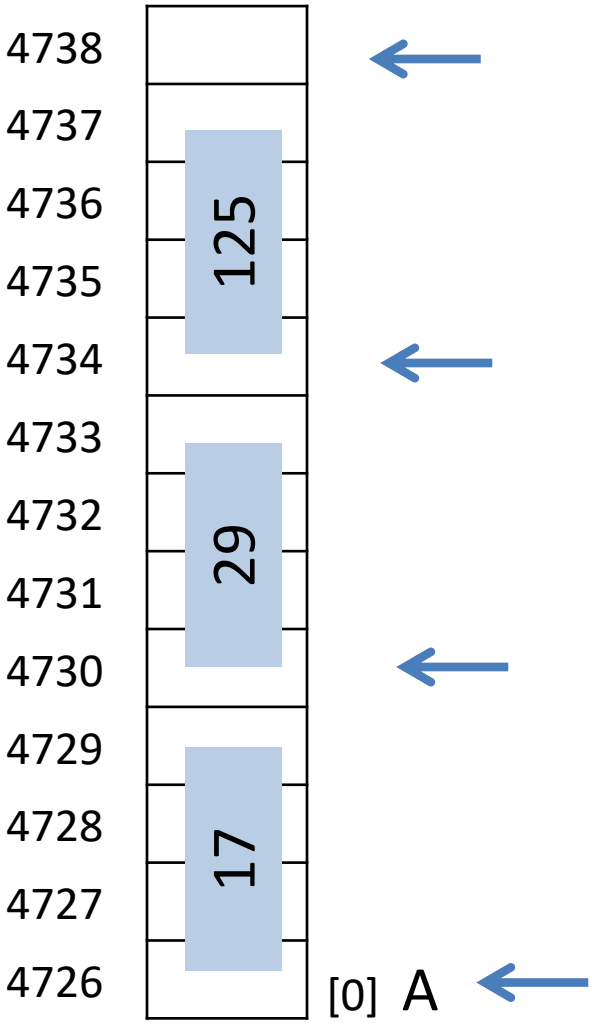
“Under the hood” look at how an array element is accessed when using the [index] method




```
int  A[50] = {17, 29, 125, …};
for (ix = 0; ix < 50; ix++)
    cout << A[ix];
```

To execute the above, EACH time through the loop the ix value is multiplied by 4 (the size of an int element) and the product is then added to the base address of the array (which is the make-believe 4726)

“Under the hood” look at how an array element is accessed when using a pointer




```
int  A[50] = {17, 29, 125, …};
int  *iPtr;
for (iPtr = &A[0]; ( ?? );  iPtr++)
    cout << *iPtr;
```

To execute the above, iPtr is first initialized to point to the base of the array (address 4726). Then, each time through loop, iPtr is incremented to point to the next int. (*which is an “under the hood” addition of 4, the size of an int, to iPtr*)